**Universidade de Aveiro**
**2010**
Departamento de Electrónica, Telecomunicações e Informática

**Marco Paulo dos Santos Fernandes**

# Arquitectura P2P e SOA para bibliotecas digitais

# P2P and SOA architecture for digital libraries

**Universidade de Aveiro**
**2010**

Departamento de Electrónica, Telecomunicações e Informática

**Marco Paulo dos Santos Fernandes**

# Arquitectura P2P e SOA para bibliotecas digitais

# P2P and SOA architecture for digital libraries

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Engenharia Informática, realizada sob a orientação científica do Doutor Joaquim Arnaldo Martins, Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Joaquim Sousa Pinto, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

Apoio financeiro da FCT e do FSE no âmbito do III Quadro Comunitário de Apoio.

**o júri**

presidente

Prof. Doutor Vítor Brás de Sequeira Amaral
Professor Catedrático da Universidade de Aveiro

Prof. Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro (orientador)

Prof. Doutora Ana Alice Rodrigues Pereira Baptista
Professora Auxiliar da Escola de Engenharia da Universidade do Minho

Prof. Doutor Osvaldo Manuel da Rocha Pacheco
Professor Auxiliar da Universidade de Aveiro

Prof. Doutor Joaquim Henriques de Sousa Pinto
Professor Auxiliar da Universidade de Aveiro (co-orientador)

Prof. Doutor José Javier Samper Zapater
Professor Contratado do Instituto Universitário de Robótica da Universidade de Valência

**agradecimentos**

O meu agradecimento para todos os que, directamente ou indirectamente, contribuiram para a realização deste trabalho, em especial:

Aos meus orientadores, Prof. Doutor Joaquim Arnaldo Martins e Prof. Doutor Joaquim Sousa Pinto, pela orientação e motivação que recebi ao longo destes últimos anos.

À Fundação para a Ciência e a Tecnologia, pelo apoio financeiro que tornou possível a realização desta tese, através da bolsa SFRH/BD/23976/2005.

À minha namorada, pelo incentivo e apoio incondicional. À minha família e aos meus amigos, pela força.

**palavras-chave** bibliotecas digitais, sistemas distribuídos, peer-to-peer, computação orientada a serviços

**resumo** Numa sociedade em que o volume e o valor da informação produzida e disseminada tem um peso cada vez maior, o papel das bibliotecas digitais assume especial relevo. O presente trabalho analisa as limitações dos actuais sistemas de gestão de bibliotecas digitais e as oportunidades criadas pelos mais recentes modelos de computação distribuída.
Deste trabalho resultou a implementação do sistema integrado para bibliotecas e arquivos digitais da Universidade de Aveiro. Este trabalho finaliza debruçando-se sobre o sistema em produção e propondo uma nova arquitectura de biblioteca digital sustentada numa infrastrutura peer-to-peer e orientada a serviços.

**keywords** digital libraries, distributed systems, peer-to-peer, service oriented computing

**abstract**

In an information-driven society where the volume and value of produced and consumed data assumes a growing importance, the role of digital libraries gains particular importance. This work analyzes the limitations in current digital library management systems and the opportunities brought by recent distributed computing models.
The result of this work is the implementation of the University of Aveiro integrated system for digital libraries and archives. It concludes by analyzing the system in production and proposing a new service oriented digital library architecture supported in a peer-to-peer infrastructure.

# Table of contents

# Index of figures

# Index of tables

# List of acronyms

API – Application programming interface
BPC - Business Process Choreography
BPEL, BPELWS – Business Process Execution Language (for Web Services)
BPMN - Business Process Management Notation
CPU – Central processing unit
DC – Dublin Core
DCMI - Dublin Core Metadata Initiative
DHT – Distributed hash table
DL – Digital library
DLMS – Digital library management system
DMFT - Distributed Management Task Force
DNS – Domain name system
FTP – File transfer protocol
GGF - Global Grid Forum
HTML – HyperText Markup Language
HTTP – HyperText transport protocol
IDE – Integrated development environment
IP – Internet protocol
ISBN - International Standard Book Number
IT – Information technology
MPEG – Moving Picture Experts Group
NAT – Network address translator
OAI-PMH – Open Archives Initiative Protocol for Metadata Harvesting
OASIS - Organization for the Advancement of Structured Information Standards
OGSA - Open Grid Services Architecture
OGSI - Open Grid Services Infrastructure
P2P – Peer-to-peer
P2PTV - Peer-to-peer television
PDF – Portable Document Format

PNRP - Peer Name Resolution Protocol

QoS – Quality of service

RCU - Central User Registry (Registo Central de Utilizadores)

REST – Representational state transfer

RPC – Remote procedure call

RSS – Really Simple Syndication

SaaS - Software as a service

SHA-# - Secure Hash Algorithm

SiNBAD - Integrated System for Digital Libraries and Archives (Sistema Integrado para Bibliotecas e Arquivos Digitais)

SOA – Service Oriented Architecture

TCP – Transmission Control Protocol

TTL – Time to live

UDDI – Universal Description Discovery Integration

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

UU - Unique electronic identity (utilizador único)

VNC - Virtual network computing

VO – Virtual organization

VRA – Visual Resources Association

W3C - World Wide Web Consortium

WS-I - Web Service Interoperability Organization

WSDL – Web Service Definition Language

WSRF - Web Services Resource Framework

XHTML – Extensible Hyper-Text Markup Language

XML – Extensible Markup Language

XPDL - XML Process Definition Language

XSLT – Extensible Stylesheet Language Transformation

# CHAPTER 1 – Introduction

In the information society we are currently living in, the volume of knowledge and information available to the public has been growing steeply. This growth can be explained by the sum of a number of factors, such as the multiplicity of dissemination media (from desktop computers to mobile devices connected to the internet), globalization and the increasing democratization of access to information and its production.

Aiming to simplify the users' task of finding relevant information within such a dense and heterogeneous volume of data, several search engines were developed, such as Altavista, Yahoo! and Google. Despite recent developments and related products which have been created alongside with these web applications in the last years, which somehow redefine the role of search websites, a search engine can be defined as:

> "Program to find answers to queries in a collection of information, which might be a library catalog or a database but is most commonly the World Wide Web. A Web search engine produces a list of 'pages'

– computer files listed on the Web – that contain the terms in a query."
[1]

Such applications function as pointers to resources, which in general are not part of the application itself. To allow users to search for those resources, search engines usually make use of two distinct entities: a web crawler and an indexing service. The first is responsible for scanning known resources and finding new ones by using the new hyperlinks found. The indexing service usually builds an inverted index of scanned resources: each word or term found occupies an entry of the index and the identifiers of resources which contain that term are then associated with it in the index. This approach allows for a quicker search of matching resources.

Traditional search engines provide however a general-purpose information retrieval. Although they allow for specific search scenarios (such as videos and images) they lack the structure and semantic knowledge of specific collections, thus treating a group of resources of unrelated matters in identical manners.

Digital libraries, on the other hand, store large amounts of well described data in a structured and well organized model and, although that is not always the case, they rely on internet. Its goal is to direct users to electronic collections, which may offer unique thematic value to researchers, historians, and general audiences.

## 1.1 DIGITAL LIBRARIES

Universities, museums, and other institutions that promote knowledge creation and dissemination, are being encouraged to build digital libraries/institutional repositories. The goal of these systems is to provide the necessary technological infrastructure to store, preserve and disseminate scientific and cultural information.

There is not a consensus regarding the definition of digital libraries. The Digital Library Federation states that:

> “Digital libraries are organizations that provide the resources, including the specialized staff, to select, structure, offer intellectual access to, interpret, distribute, preserve the integrity of, and ensure the persistence over time of collections of digital works so that they are readily and economically available for use by a defined community or set of communities." [2]

The DLib Working Group on Digital Library Metrics defined a digital library in a different manner:

> “The collection of services and information objects that support users in dealing with information, and the organization and presentation of those objects, available directly or indirectly via electronic means.” [3]

Another common approach to define digital libraries is to use the traditional library metaphor [4], comparing the provided services – information access, search methodologies – in both scenarios. In that sense, digital libraries almost appear as an natural evolution, in which there is a similar paradigm with enhanced functionalities (full text search, bookmarking, annotation, etc.).

From the above definitions, we summarize the various concepts into the digital library definition we will use from this point forward:

> An information system which provides online search, selection, and dissemination of structured collections of digital services and objects (globally known as resources), and promotes the preservation and integrity of those resources.

It should be noted that we employ the term digital library in its more broad definition, comprising digital archives, museums, and every similar system. Digital archives, for instance, differ from digital libraries (in its strict definition) in the sources of information (primary/unedited instead of secondary), organization of information (categorically rather than individually), and preservation (a primary concern in archives). We will not make such a distinction.

## 1.2 DIGITAL LIBRARIES EVOLUTION

Surprisingly, some of the concepts behind digital libraries such as preservation have been present for more than a century. Microfilm technology, a compact storage medium for paper documents, is reported to have been first used in 1870 during the Franco-Prussian War [5]. Later in the 1930s, when World War II threatened to destroy the archive of the British Museum, University Microfilms started the preservation of printed works on microfilm.

In 1945, Vannevar Bush's [6] proposed a system called memex, where ultra high resolution microfilm reels were coupled to multiple cameras by electromechanical controls. The prophetic essay also introduced a concept similar to hypertext.

The first remotely accessible databases came online in the late 1960s. These early databases mainly dealt with legal, scientific, and government information [7]. CD-ROM and local databases appeared in the mid-1980s, allowing images to be stored and retrieved.

In 1989, Tim Berners-Lee proposed a global and distributed hypertext information exchange network, which would become the HTML (Hyper-Text Markup Language) based internet [8].

In 1994, the Library of Congress announced a National Digital Library, and Libraries Initiative, a research effort involving several universities in the study of digital libraries [7].

In 1995, Kahn and Wilensky [9] defined an architecture of distributed digital objects services. According to the authors, a digital library belongs to such a category: it is a repository of digital documents, properly and uniquely identified, and information about those objects, named metadata. Later, in 1997 [10] an architecture for digital libraries was presented with four main components: digital objects, identifiers, repository, and user interfaces.

With the maturing of the involved technologies, current digital libraries face more challenges outside the technical scope, namely copyright and legal issues [11].

### *1.2.1 Technology impact*

Internet growth and its degree of adherence is the single most important factor in the evolution of technology in information systems. It has become the favored media of production and dissemination of information. Millions of users connect daily to a network with more than half a billion hosts [12].

The need for document preservation, along with internet and the evolvement of desktop software and hardware, have ignited a quest for mass digitization of historic material: printed (books, letters, etc.), photographic (photos, posters), video (VHS and Beta) and audio (vinyl).

On the other hand, it has accomplished a dramatic shift in how society functions. For instance, many private institutional and commercial publications are no longer created in paper – only electronic versions are produced. Companies and individuals are starting to rely solely on digital invoices, reports, and correspondence.

While it seems clear that having all this digital material makes it easier to access and distribute information, it also points that efficient and easy to use information management software is crucial. Without one, searching for a document in a repository with millions of files becomes little different from looking for a piece of paper in a stack of documents.

### *1.2.2 Technology evolution*

The first digital libraries, as the generality of information systems, were monolithic applications which used proprietary data and description rules. With the evolution of internet, researchers, librarians and software architects found the need for a standardization of information and protocols to simplify communication between systems and ease the understanding of external data.

XML (Extensible Markup Language) has been the de facto standard for describing and transmitting data for some years. It provides a text-based language whose main purpose is to facilitate the sharing of data across applications, platforms, institutions, etc. Due to its flexible and customizable nature, XML has been the skeleton for numerous standards, such as SOAP, WSDL (Web Service Definition Language), XHTML (Extensible HyperText Markup Language), RSS (Really Simple Syndication), and technologies such as Web Services, OAI-PMH (Open Archive Initiative Protocol for Metadata Harvesting), and BPEL (Business Process Execution Language).

Web Services provide a standard and interoperable means of machine to machine interaction, using a well know interface, based on SOAP and WSDL. Web Services allow the transparent communication of machines from different programming languages, platforms and operating systems. They also allow the aggregation and consuming of information in a simple way. Despite the standard interaction, there is no standard for the data structures being passed. Even in the case in which two digital libraries store data with the exact same schema and metadata, each system does not have a priori knowledge on how to access information on the other: which remote methods to invoke, what data structures are provided, etc.

Open access and open archives initiatives have become popular in the last years. The underlying philosophy in these initiatives is the availability of digital content free of charge. It commonly embraces the concept of self-archiving, by which researches make available their own work. Particularly important for the interoperability between digital libraries is the OAI-PMH [13] protocol (Open Archives Initiative Protocol for Metadata Harvesting). This HTTP based protocol defines how a data provider exposes its metadata to harvesters (other digital libraries, federation sites, etc.) by using clearly defined XML structures, thus eliminating the problem of a priori knowledge.

## 1.3 MOTIVATION

Current digital libraries face new challenges and demands. With the opportunities given by the Internet, these information systems must be able to deliver very high amounts of data to a growing number of users. Also, from our digital libraries definition, such information systems must not act only as data repositories – they should provide services for both humans and machines.

The centralized model, in which a server not only hosts the web site but is also responsible for all the underlying tasks required by the digital library, therefore lacks the necessary scalability and flexibility. A distributed approach, which promotes interoperability and cooperation, is a key element for success.

### 1.3.1 SInBAD

Beginning in 2004, the author was an active member of the conception and development of SInBAD, the integrated system for the digital library and digital archive from the University of Aveiro. It is composed by a number of heterogeneous collections, such as photographs, books, articles, and videos.

In the scope of this project, a number of issues had to be addressed:

1. Metadata must be uniformly described using standards, which is not a trivial task due to the heterogeneity of resources;
2. Instead of being an isolated system, SInBAD must be able to interoperate with other systems in the institution such as the scientific bibliographic archive or library's bibliographic application;
3. Even small or medium sized organizations can produce very large amounts of data and metadata, both of which must me consistently stored, secured, and backed up – the system should be able to handle such volumes of data without degrading user experience.

The objective of this doctoral thesis was to design and implement the SInBAD digital library and, using it as a first conceptual and working basis, to study new computational models, such as Peer-to-Peer and Service Oriented

Architectures, and how they can provide the skeleton for better distributed digital libraries. This work should result in a digital library architecture which allows:

- To collaboratively store very high volumes of data. Digital libraries typically store large amounts of information, which a decentralized approach can more properly accommodate;
- To create a distributed service overlay. More than a simple repository, a digital library depends on a number of – possibly time consuming – services to its normal functioning. Distributing the execution of those services can greatly improve the performance and responsiveness of systems;
- Standardize interoperation between systems and components, and consume services in workflows which are flexible and dynamic;
- To efficiently search distributed resources. As data and services become decentralized, it is crucial to have efficient mechanisms to find these resources.

## 1.4 NEW COMPUTATIONAL MODELS

### 1.4.1 Peer-to-peer

New computational models and protocols have been proposed to create more scalable, interoperable and robust systems. One such model is Peer-to-peer (P2P), which is radically different from the classic/server architecture. In P2P, each network node acts both as server and client, producer and consumer. Numerous advantages derive from this approach [14]:

- It can operate at the edges of the Internet, behind firewalls and NAT (network address translator) systems;
- It supports highly transient connections;
- It can take advantage of unused resources of connected nodes;

Current P2P applications are capable of creating network overlays which connect millions of users with a virtually unlimited data volume. Also, by using a decentralized architecture, P2P does not have specific (central) points of failure

which can break an application or significantly reduce its performance. To provide even greater redundancy, some solutions apply replication of data and metadata between peers.

P2P has been traditionally associated with file-sharing applications, such as Kazaa and Napster, in which each user (node) can share its files, search for and download other resources. We believe that its numerous advantages make it attractive for the implementation of more complex systems, such as a digital library.

### *1.4.2 Grid*

The Grid model refers to an infrastructure which allows the integration of computers (usually dedicated), networks (high bandwidth), information and other resources (CPU cycles, memory, etc.) of several organizations in a cooperative manner. Such integration is accomplished through a distributed system which allows searching, aggregating, and selecting geographically disperse resources [15].

Grid computing, which originated from the need to efficiently solve computationally intensive tasks, distinguishes from other distributed applications for being oriented to the resolution of complex and demanding problems, traditionally scientific and multi-institutional.

Each group of organizations and/or individuals which share resources based on a set of common rules is usually called a “virtual organization” (VO). Taking advantage of a Grid environment requires using specific software with certain requisites [16].

#### *1.4.2.1 Standards*

Although there are numerous individual Grid projects and emerging standards [17], one of the challenges has been trying to find an international consensus on which global standards to adopt to make these autonomous and independent projects to interact in a larger Grid. The exception is GridFTP, a file transfer protocol defined within the Globus Toolkit (see section 2.4.1).

Standard bodies include the Global Grid Forum (GGF), the Organization for the Advancement of Structured Information Standards (OASIS), the World Wide Web Consortium (W3C), the Distributed Management Task Force (DMTF), the Web Services Interoperability Organization (WS-I), and groups within Internet2.

GGF, which is the primary standards-setting body, promoted the OGSA (Open Grid Services Architecture), which aims to define a common service-oriented architecture for Grid-based applications.

The first instantiation of the OGSA architecture resulted in OGSI (Open Grid Services Infrastructure), based on the concept of Grid Services, which represented a modified version of Web Services that supported state management (unlike the standard, stateless, Web Services). Growing dissatisfaction and criticism towards OGSI, due to the extent of the specification and the need to use modified WSDL descriptions, led to the development of a new infrastructure: the Web Services Resource Framework (WSRF). Unlike OGSI, WSRF is based on unmodified Web Services specifications. OGSI is now considered obsolete.

### *1.4.3 Grid vs. P2P*

Some authors argue that Grid computing is essentially a P2P system with distinct implementation details and that, in the future, both concepts will become one. Such synergy, predicted and desired by many [18][19], and which may speed up the development of both study areas, is based on the similarities in both paradigms. So far, however, they both still present some distinct characteristics:

- Decentralization – Although it promotes resource decentralization, Grid computing always performs some form of centralization in a reduced number of computers; P2P, on the other hand, allows a complete decentralization and treats all peers as equals. This makes P2P more scalable and failure resilient, although it raises discovery and search implementation issues.
- Security – Security plays an important role in Grid, and a great deal of importance is given to authentication, authorization, and integrity; in P2P,

mostly due its file-sharing origins, generally only a few basic security and integrity mechanisms are implemented, if any.

- Connection – Grid connection is typically too rigid to accommodate a simple and dynamic connection of new nodes in the network; P2P allows very dynamic connections and disconnections to the network.
- Services – While one of the Grid motivations is to allow the remote invocation of resource attribution and task execution services, there is no such mechanism in traditional P2P, although information transfer protocols are well developed.
- Discovery – Nodes and resources information in Grid are stored in a centralized fashion which allows them to be easily found; P2P deals with more dynamic scenarios and promotes self organization, making resource discovery a dynamic procedure.
- Fault tolerance – The nature of Grid computing demands the existence of some sort of fault tolerance mechanisms, although this remains somewhat rudimentary; although most P2P do not have sophisticated fault tolerance mechanisms, its decentralized nature reduces this problem.
- Standards – While Grid applications generally adopt standard interfaces, representation schemes, and communication patterns, most P2P environments still use proprietary protocols.
- Usage – Grid networks are usually composed by stable and homogeneous nodes from closed communities; it aims to solve problems too complex to execute in a timely fashion on a single computer. P2P tend to favor open communities with anonymous users and unpredictable behaviors.

### *1.4.4 Service Oriented Architectures*

Traditionally, applications were built in an isolated and closed environment. Even if such applications are designed in a modular fashion, its components and methods are only known by and available to the application itself.

Service Oriented Architectures (SOA) refers to a conceptual model in which a business process is made available as a loosely coupled service. SOA evolves from both the distributed computing concept (services can be and usually are

consumed from remote machines) and modular programming (its units/services commonly aggregate related functions).

There are several definitions available for the Service Oriented Architectures model:

> "SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains." [20]
>
> SOA is a computer system's architectural style for creating and using business processes, packaged as services, throughout their lifecycle. [21]
>
> "SOA is a business-centric IT architectural approach that supports integrating your business as linked, repeatable business tasks, or services" [22]
>
> "SOA is the organizational and technical framework that enables an enterprise to deliver self-describing, platform-independent business functionality and make it available as building blocks of current and future applications." [23]

It is worth mentioning that none of these definitions is bound to any specific technology. SOA refers to an abstract model which can be implemented by using many different frameworks and platforms.

Although being usually associated with benefits for large enterprises, the SOA approach is a shift in design and style of software which can bring numerous advantages to small companies and organizations as well.

Some of advantages identified [24] are:

- Allows creating new business value from existing data. New services can use data federation from different databases to create a new view of information.

- Creates and abstraction layer, in which each service only needs to know about how it implements its functionality and makes it responsible for its own data.
- Facilitates software maintenance. SOA promotes the creation of basic services (such as for data access) highly focused on a specific need. Applications can then be built using these simple services as well as composite services. This service modularity makes it easier to maintain, update and redesign existing functionalities.
- Enables service marketplaces. By using composite services as the application's building blocks, basic services can be consumed from external and dedicated service marketplaces. These businesses have the advantage of making administration of contracts to service providers more streamlined and uniform and providing a service registry to help finding services, help users sharing problems and solutions.

## *1.5 RESULTS*

The primary result of this work is the conception and implementation of the University of Aveiro digital library and archive, of which most modules have been in production since 2005. This system has also become the entry point for the University digital repository to external researchers, historians, and generic users.

The finalized system successfully responds to the goals set, such as distributed architecture, flexible description models, high granularity, high interoperability, and modularity. As will be shown throughout this work, and unlike popular digital library and archive systems, SInBAD was designed so that its components could be distributed – subsystems, services, and data. It also provides a higher description and search granularity, and seemingly integrates heterogeonous data. As a result, the author has published a book chapter [25] and four scientific articles related to the system [26][27][28][29].

To empower SInBAD with the ability to distribute data and workload to other network nodes, it was built on top a distributed system also conceived in this work - DisQS. Results show the system successfully scales and has a modular and

service oriented model which provides a more flexible and dynamic architecture. Three articles were published regarding that system alone [30][31][32].

The culminating of this work's research is the proposed architecture for digital libraries based in peer-to-peer technologies and service oriented computing. The architecture is designed to allow services and data (treated as generic "resources") to be distributed through a network, to achieve a greater flexibility to discover services, and to optimize the execution of the business processes.

Two publications were made describing the proposed distributed architecture [33][34]. Also, the preliminary analysis of the state of the art, the study of similar technologies, and the conception of the architecture several articles originated several published articles related to resource integration and aggregation [35][36], peer-to-peer networks [37], grid computing [38] and search engines [39].

## 1.6 DOCUMENT OUTLINE

The rest of this document is structured as follows. CHAPTER 2 overviews the relevant state of the art in digital library management systems (DLMS) and the new computational models applicable to the conception and development of digital libraries, namely Peer-to-peer, Grid computing and service oriented architectures.

As a consequence of the limitations found in existing DLMS, CHAPTER 3 describes the design and conception methodology used for the development of SInBAD, and discusses the adopted architecture. This architecture is based on a distributed system conceived to essentially take advantage of data storage capabilities in remote machines. Such storage is made according to a comprehensive metadata analysis of several standards suited for the very heterogeneous repository. The devised system is also extensively based on both internal and external services.

CHAPTER 4 follows the discussions made in the last section of the previous chapter, namely regarding the possible improvements in the system, and analyses a group of contributions in the scope of service orientation, business process execution, and peer-to-peer networks. The proposal presents a service layer on

top of a peer-to-peer infrastructure which allows services to be discovered and invoked within such networks even when there is low or inexistent connectivity between consumer and provider. It also offers insights on the improvements of business process execution when based on such infrastructure.

Finally, CHAPTER 5 presents the conclusions of the work and discusses possible directions for future work.

# CHAPTER 2 – State of the art

## 2.1 INTRODUCTION

In this chapter an overview of existing digital library management systems is made. The following sections review the state of the art of tools, systems and frameworks regarding P2P, Grid and service oriented computing.

Regarding the general conception of digital libraries the work of DELOS is of particular interest. Funded by the European Union's Sixth Framework Programme, DELOS is a network working for the excellence in digital libraries. It is formed by a number of workgroups spread throughout Europe.

In [40], each workgroup contributed with its vision of a digital library architecture. Most contributions point towards the use of P2P, Grid and SOA concepts in the infrastructure of future systems. We highlight some contributions in the next sections.

### 2.1.1 University of Athens (Greece):

With the increase of the volume of available information, the size of future digital libraries should lead to the adoption of federated databases or ones based on the Grid or P2P paradigm. Regardless of the progresses made in hardware, distributed architectures are seen as the only solution to scalability issues.

### 2.1.2 Eidgenössische Technische Hochschule Zürich (Switzerland):

The future digital library should be highly scalable, customizable and with an adaptive infrastructure. To accomplish such goals, that infrastructure should use a combination of P2P (loosely coupled service integration, information sharing), Grid (dynamic allocation and deployment of complex and computationally intensive services) and SOA (definition of the semantics and usage of services).

Figure 2.1 (copied from the reference) depicts an example of how the services available in a network are used in the *Insert Image* process.

### 2.1.3 Istituto di Scienza e Tecnologie (Italy) and Fraunhofer-Gesellschaft Institute (Germany):

The Istituto di Scienza e Tecnologie – Consiglio Nazionale delle Ricerche and the Fraunhofer-Gesellschaft Integrated Publication and Information Systems Institute workgroup places some focus on the need to create virtual organizations, composed by distributed individuals working together in a temporary basis.

From the infrastructure point of view, the workgroup aims to create an architectural framework composed by three elements:

- The technical infrastructure responsible for supporting basic functionalities such as dynamic resource allocation, sharing, security, or QoS;
- A set of services which implement the typical digital library functionalities;
- Application specific services which provide access to shared repositories or tools and comply with the infrastructure rules.

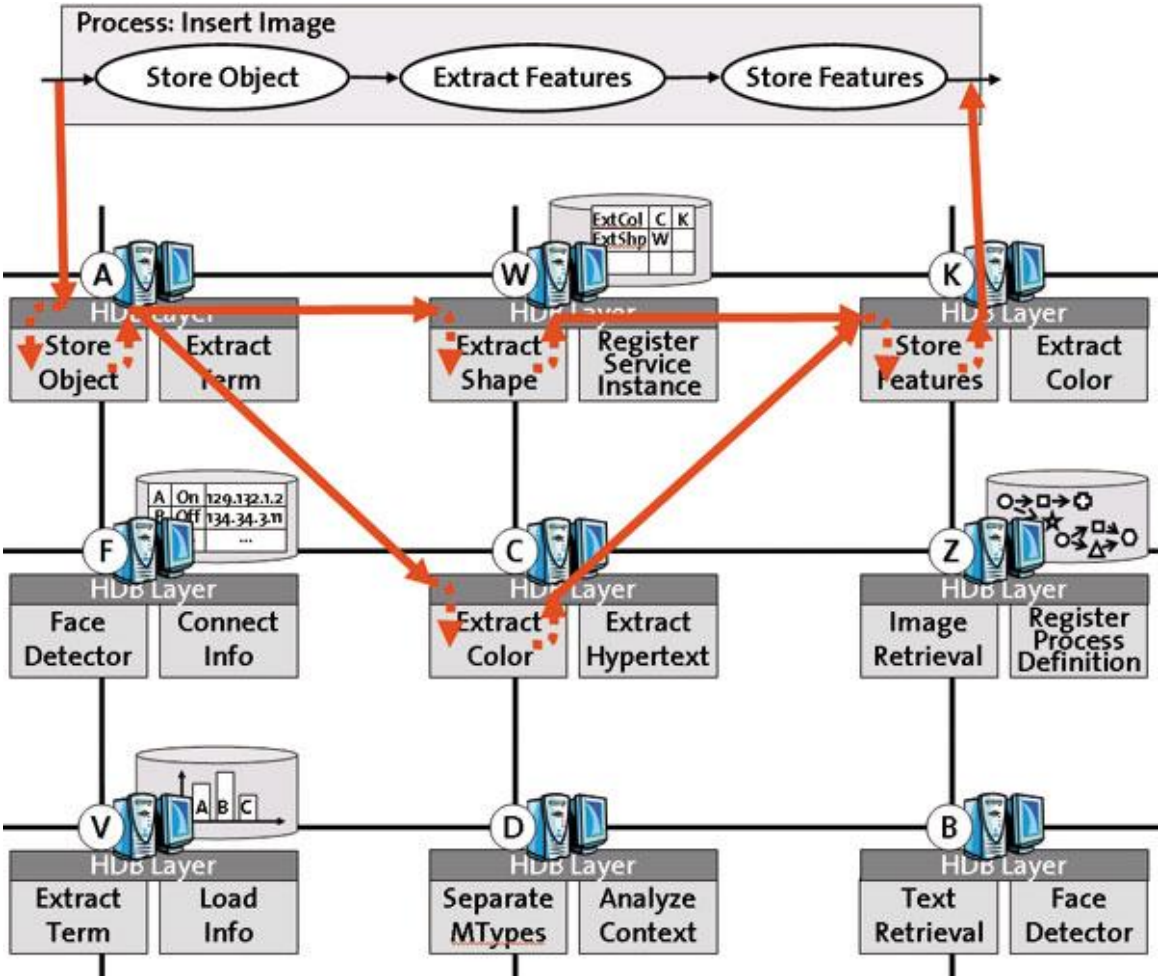


**Figure 2.1 – Digital library process decentralized execution**

### *2.1.4 Kuratorium OFFIS e.V. (Germany):*

The research focus is made on super peer networks, in which nodes are chosen to form a hierarchical network. Super nodes maintain metadata indexes of available resources and allow combining the efficiency of centralized client-server model with the autonomy, load balancing, and robustness of distributed solutions. It also permits implementing distinct protocols and rules within each cluster.

## 2.2 DIGITAL LIBRARY MANAGEMENT SYSTEMS

### 2.2.1 DSpace

DSpace [41], one of the most popular DLMS in archives and universities, is an open-source system developed by HP and MIT which acts mainly as a repository for educational and scientific material produced by an organization or institution. DSpace is able to store virtually any type of document, which is described using the Dublin Core Metadata Initiative (DCMI) [42] metadata set and exposed to external entities through an OAI-PMH interface, thus promoting interoperability.

The DSpace system is organized into three layers (Figure 2.2, available at the MIT website): the storage layer is responsible for the physical storage of data and metadata; the business logic layer handles the management of archive, its users, authorization, and workflow; the application layer contains components for the communication with other applications.

DSpace has some limitations which reduce its applicability in more complex digital libraries, such as:

- The lack of restriction in the access to documents (or parts of it) disregards copyright issues;
- The use of a single repository reduces its scalability and error resilience;
- A rigid description model reduces cataloguing and indexing flexibility;
- Search granularity is limited to a complete document and only the Dublin Core descriptors can be searched;
- At the time of writing, Web Services for DSpace were still under development by MIT;
- An authority database, which maintains information about the authors and links them to the records, is inexistent;
- Complete installation and configuration of a DSpace repository may take several weeks [43].

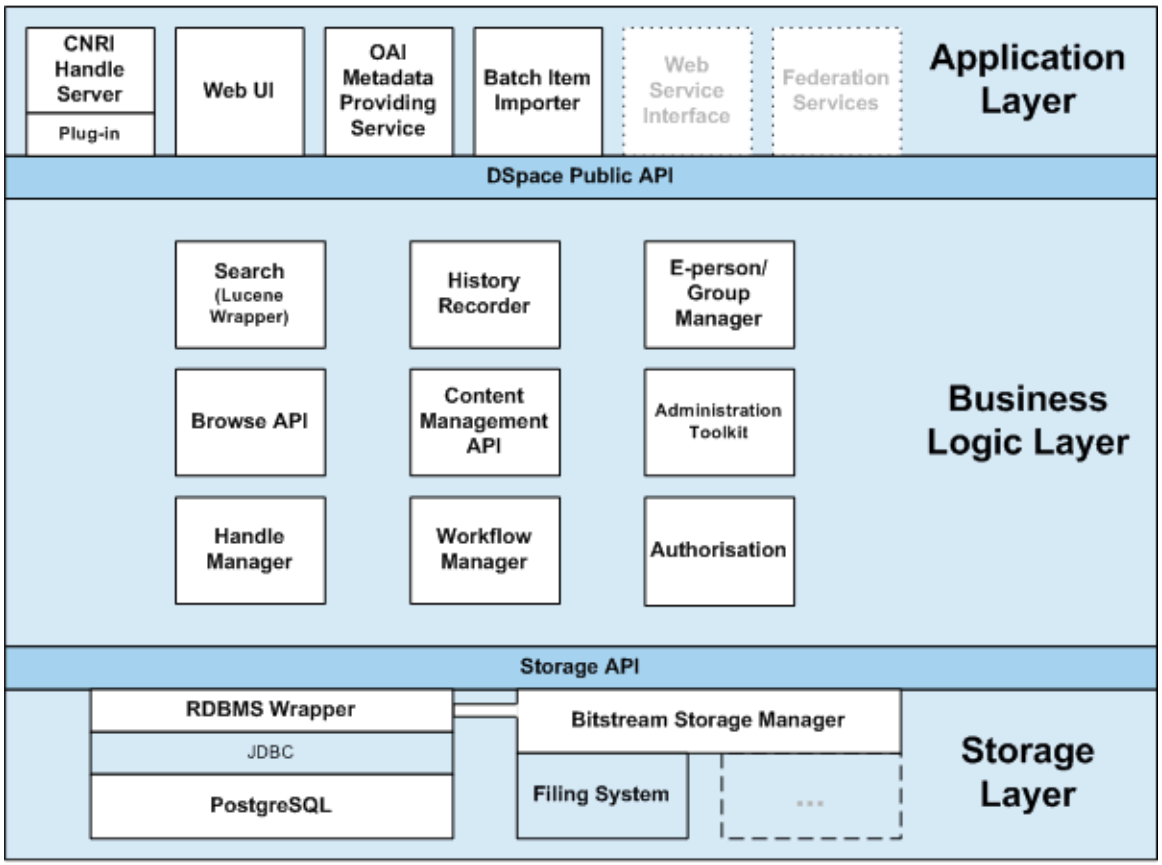


**Figure 2.2 – DSpace architecture**

### *2.2.2 EPrints*

EPrints [44] is Linux based software for the deployment of a generic web-based open-source institutional repository developed by the University of Southampton. It is mainly intended to create open archives of research papers, although it can be adapted to store any digital file.

EPrints has identical features to DSpace. It supports self-archiving, OAI-PHM, and is flexible enough to store any file type. It has also basically the same limitations: the lack of a granular document control, centralized approach, and long installation and configuration.

### 2.2.3 Fedora

Fedora [45] is a Java based open source framework for the management and delivery of digital content developed jointly by Cornell University Information Science and the University of Virginia Library.

It is Web Services based, supports distributed repositories, and is programming language agnostic. It also has REST (Representational State Transfer) APIs and an OAI-PMH provider. One of its most interesting features is its plugability for the storage mechanism: instead of using the (default) file-based storage, one can develop a custom plug-in or use existing ones, such as the Amazon S3 service or the iRods plug-in, which allow data to be stored in Amazon's data storage or in an iRods installation.

It is not, however, a complete and ready to use system, but instead a repository system with Web Service interfaces. It is reportedly complex to use [46].

### 2.2.4 Greenstone

Greenstone [47] is an open source software suite which allows creating collections for digital libraries produced by the University of Waikato in cooperation with UNESCO and the Human Info NGO. It is flexible and supports several media formats. Data is composed of resources (the digital objects) and documents (the metadata).

Greenstone may be distributed by using Agents, which use SOAP messages through a Message Router to accomplish tasks.

Greenstone not only has an OAI-PHM provider but can also import records from an external OAI-PHM repository. Data can also be imported from and exported to a DSpace repository.

A severe limitation of the system lies in the fact that indexes must be rebuilt each time the repository is updated, which means Greenstone is more suitable for static (or semi-static) collections.

### *2.2.5 BRICKS*

The BRICKS Project – Building Resources for Integrated Cultural Knowledge Services – is a European Commission funded research project aiming the creation of a cultural heritage network [48]. The BRICKS Community is a worldwide federation of cultural heritage institutions, research organizations, and technological providers.

Its approach is based in the decentralization of resources, to increase error resilience, scalability and reduce maintenance costs. Such decentralization is obtained using a P2P layer implemented with P-Grid (see section 2.3.5.6).

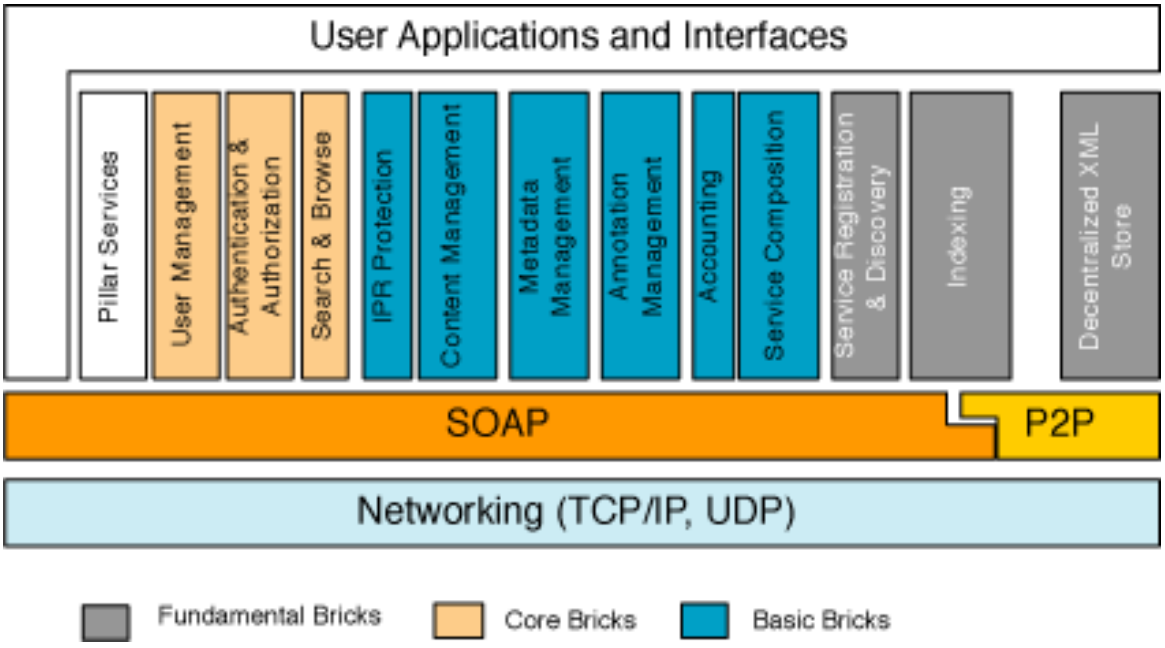


**Figure 2.3 – BRICKS architecture**

The infrastructure also relies on a SOAP module upon which services are built. Figure 2.3 (from the BRICKS website) depicts the architecture of a BRICKS node (called BNodes).

## 2.3 PEER-TO-PEER

### 2.3.1 Introduction

A Peer-to-peer (P2P) application is a networked system whose architecture does not rely on dedicated servers. Instead, each network node (the peers) act as both client and server – thus becoming responsible for its own resources – and communication is established between multiple nodes. P2P networks are usually simpler than those from traditional client/server, although they introduce a number of issues regarding performance, management, and availability.

Implementing P2P systems usually involve the use of a P2P network overlay, an abstract layer which transparently and independently of the physical network deals with connectivity, addressing, and communication. To the upper layers, this overlay acts as a messaging channel, in which only notification (connect or disconnect, ping, topology change, etc.) and search (query and query response) messages are exchanged. To actually transfer resources between peers, communication is usually accomplished by using a different protocol, such as HTTP.

In summary, a P2P system adopts three principles:

- Resources are shared (files, services, disk space, bandwidth)
- Resources are decentralized, which derives from the fact that each peer manages its own (local) resources
- The network is self-organizable: since there is no central entity to coordinate the nodes, peers self organize in an autonomous fashion by using pre-established behaviors.

### 2.3.2 Common features and issues

Most popular P2P based applications aim the anonymous sharing of files between users. However, P2P can help solving the scalability issues inherent of centralized solutions in many different scenarios. In this section an overview is made of the basic characteristics common to a large majority of P2P applications.

#### *2.3.2.1 Binding*

When a user accesses a website on the internet, an early-binding takes place: a DNS server performs a static translation between a name and an IP address. In practice, most sites have long-term internet connections with the same IP address, and therefore the early-binding mechanism performs reasonably well.

In the case of modern P2P systems, however, nodes may belong to a wide range of mobile devices, have dynamic addresses, be placed behind network address translators, use different protocols, etc. In this scenario, there occurs the dynamic translation between names and physical addresses: late-binding.

#### *2.3.2.2 Scalability*

P2P systems can be extremely dynamic in size, adopted topology and network activity. To allow for a high quality of service (QoS), P2P applications should tackle issues such as high load, network congestion, appearance of hotspots (peers with very popular resources), among others. To properly tackle possible problems, some systems employ mechanisms for caching, replication and homogeneous load balancing.

#### *2.3.2.3 Failure resilience*

In most cases, thanks to the decentralization of control and coordination, P2P can be more resilient to hardware or software failures. Nevertheless, since some peers are more relevant to the network than others, the failure of certain nodes can be troublesome even in decentralized environments.

#### *2.3.2.4 Security*

Although the first P2P systems did not adopt more than trivial security mechanisms, the P2P community has been gradually paying more attention to this topic. Attacks to a P2P system usually make use of the knowledge of the adopted topology. In hybrid P2P, where some form of centralization is used, attacks aim the central peers. In completely decentralized topologies, targets are typically the most popular nodes.

P2P may suffer attacks which are similar to those perpetrated to centralized applications, such as denial of service. The most common attacks are however performed from inside the network when:

- Peers provide resources that do not match the description (for instance, the Recording Industry Association of America reportedly distributed fake audio files in popular P2P networks to disencourage users from illegally downloading music);
- Peers distribute corrupted resources;
- Peers act as “leechers” and do not contribute with resources, they only consume others’.

#### *2.3.2.5 Anonymity*

Since the P2P concept became popular in file sharing applications, providing an anonymous access to the network has always been a matter of concern. Some of the most sophisticated programs implement anonymity for both the peers and the queries.

### *2.3.3 P2P Topology*

In this and in the next sections existing P2P topologies and data structures are analyzed in the scope of digital libraries. It is worth noting that such analysis could differ within a different domain area. The following discussion would be different if the intended application scope was that of an instant messaging, for instance.

Regarding the network topology P2P systems can be classified with one of four main categories: centralized, decentralized, hierarchical, or hybrid [14].

#### *2.3.3.1 Centralized*

In centralized P2P systems (Figure 2.4), such as Napster, nodes connect to the network by registering themselves at a central server and sending an index of the resources they maintain. When a node wishes to find resources, it sends search queries to the server, which looks up its global index to retrieve matching

items. The actual file transfers are performed between the peers without the intervention of the server.

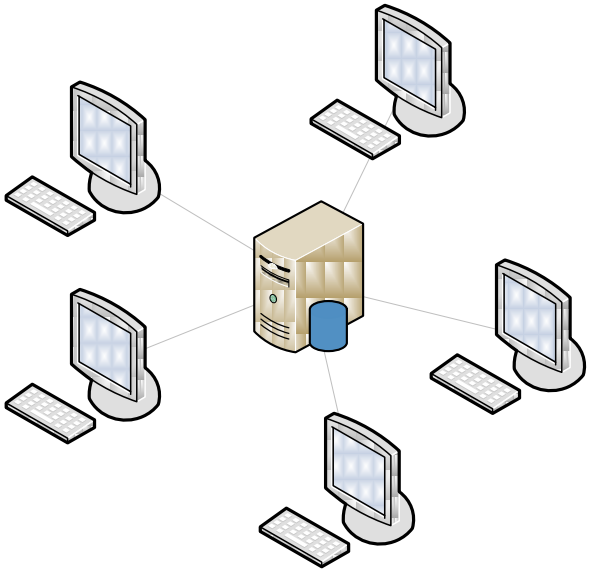

**Figure 2.4 - Centralized P2P topology**

Although centralized based P2P systems are bandwidth-efficient and easier to administer, such systems cannot scale as much as decentralized ones due to the bandwidth and processing power limitations of the server. More crucial than this, if the server becomes temporarily unavailable, the entire network ceases to work properly.

#### *2.3.3.2 Decentralized*

Completely decentralized (or pure) P2P systems (Figure 2.5), such as Gnutella 0.4, are based in the inexistence of structure or hierarchy. All peers remain equal among each other throughout their life-cycle. To enter a network, new nodes connect to any known peer and become neighbors of a small set of peers. When a search is made in a peer, a query package is broadcasted to the connected neighbors with a fixed time to live (TTL). Decentralized P2P networks are also generally self-organized, hence they adapt themselves dynamically.

Pure P2P systems can grow up to millions of connected users without significantly degrading performance but cannot properly scale. While a search query performed in a P2P network composed by only a few hundred nodes could eventually find every matching resource, this no longer remains true in much larger networks due to the TTL. From our point of view, this fact alone is sufficient to not implement digital libraries in pure P2P.

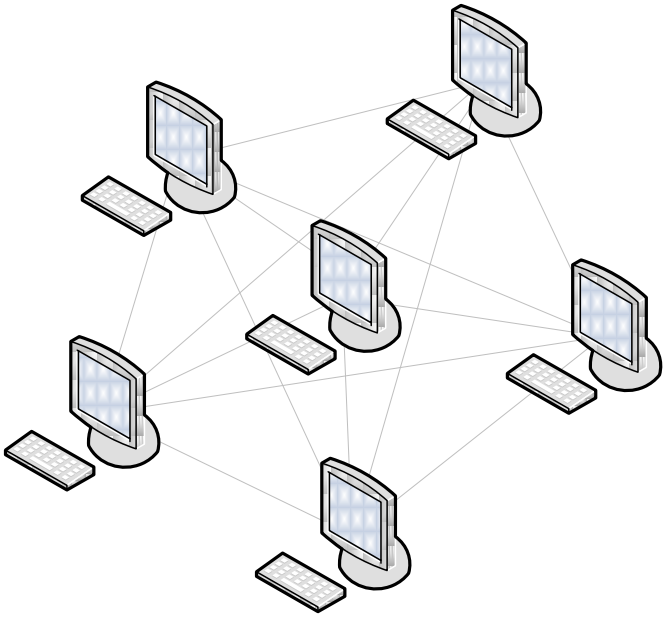

**Figure 2.5 - Decentralized P2P topology**

#### *2.3.3.3 Hierarchical*

A hierarchical topology usually follows an underlying structure: social, geographical, etc. In this topology, nodes connect to the network in a predefined level of the tree. Indexes of the metadata can be stored only in each node or parent nodes may aggregate the indexes from all its child nodes.

Such type of network has the advantage of mimicking a known and logical structure, which makes it easier to find information based on locality.

#### *2.3.3.4 Hybrid*

Most modern P2P applications apply some sort of hybrid topology, aiming to achieve a robust network solution by combining characteristics of other topologies. To implement our framework the chosen topology relies on the concept of super peers [49][50] – peers that act as an interface between a cluster of peers and the rest of the network. This allows combining the robustness of centralized solutions with the flexibility and scalability of decentralized ones.

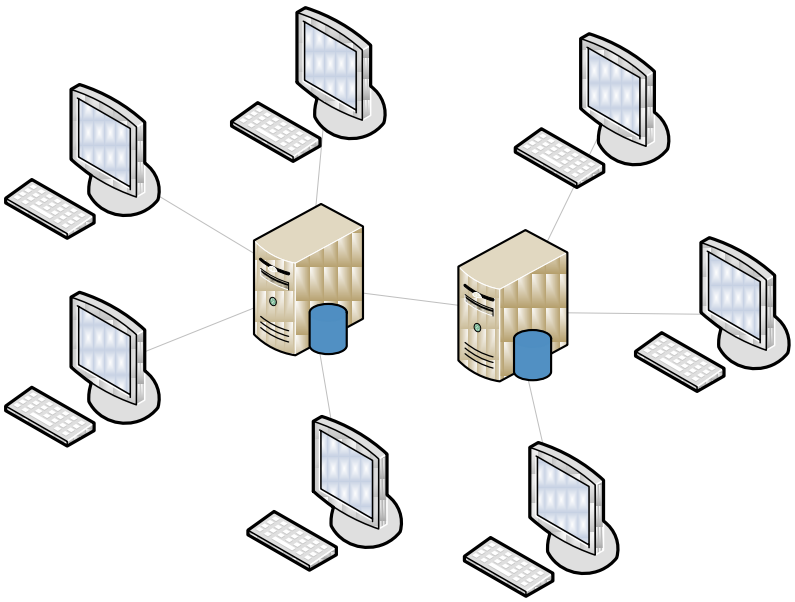

**Figure 2.6 – Hybrid P2P topology**

Super peers, however, only reduce the number of peers to query by some order of magnitude. In very large scale scenarios, its behavior becomes identical to that of a decentralized topology. To solve that problem, super-nodes can be arranged in a hierarchical tree-like topology which follows a geographical or organizational model. Large organizations can have super peers distributed according to geographic locations and different organizations can collaborate to the same digital library by becoming a tree branch of the same tree.

This has many advantages: queries can be adjusted according to the hierarchy; different rules may be set in each organization/tree branch; indexing can optionally be hierarchical – searches can be made in an entire branch by querying the root.

### *2.3.4 Data structure*

Regardless of the topology chosen, which defines how nodes connect themselves, one must decide how to actually populate peers with data. P2P systems usually take one of two basic approaches: structured or unstructured.

Mischke and Stiller [51] analyzed the problem of distributed searches in different structural data space designs.

#### *2.3.4.1 Structured network*

Structured networks such as Chord [52] or CAN [53] rely on distributed hash tables (DHT) – a class of decentralized systems which provide lookup mechanisms – to retrieve the network location (current or to be) of a file.

The most common approach consists of conceptualizing a grid-like data space (the key space). Upon entering the network, peers are assigned one (or more) of the grid cells (usually by hashing their own identifiers), and they become responsible for all the data mapped into those cells. Resources are mapped into keys by hashing one or more descriptors into the key space. Usually, the hashing mechanism allows an efficient routing mechanism, since each node can redirect requests to the neighbors whose key is closer to the query hash.

Structured P2P networks are highly scalable and rely on the fact that there is a metric for a peer to quickly retrieve any resource by using the mapping function. Also, redundancy (and load balancing) can be achieved in a simple manner by assigning two or more peers to the same key space.

Its main disadvantage is the fact that searching by metadata is a complex task which may require broadcasting queries to the network. Although solutions based on metadata summary propagation have been developed [54], they do not

provide satisfactory search capabilities for digital libraries. Also, in very dynamic scenarios, peers leaving and entering the network require intense computation and communication to maintain the network properly structured.

#### *2.3.4.2 Unstructured network*

Unstructured networks, on the other hand, have no predefined strict rules for storing data. Resources are initially stored in its originating peers and can be later replicated according to the protocol rules.

These classes of P2P networks are ideal in very dynamic networks, where constantly updating a hash table can be troublesome.

Unstructured networks scale worst than DHT based ones and may generate larger network traffic in some situations. However, its flexibility makes it more attractive to digital libraries and is the chosen data model for our framework. It does have the limitations of the structured model and, since each node is responsible for its own data, queries can be as complex as desired – each node will answer with the best result possible.

### *2.3.5 File sharing*

The traditional application scope for P2P is file-sharing. In this section, we outline some of the most popular file-sharing protocols and applications.

#### *2.3.5.1 Gnutella*

Gnutella is one of the most popular file-sharing P2P protocols. It is used and supported by applications such as LimeWire, Shareaza and iMesh. The now outdated 0.4 version of the Gnutella protocol [55][56] operates on a purely decentralized fashion. To enter the network, a node must connect to an already connected peer. In order to find resources, a search query is broadcasted to all directly connected peers, which in turn retransmit it to their neighbors. Since queries are “blindly” sent, network packages include a time-to-live (TTL) field to avoid the perpetual retransmission of messages. The actual transfer of files is accomplished by using HTTP.

From the user point of view, applications running Gnutella may offer satisfactory results, mainly because of its volume of data. Gnutella is also generally tolerant to network failures and can easily adapt to highly dynamic environments. This protocol has however several limitations: it promotes the flooding of messages; it does not guarantee that all nodes can be reached (due the TTL); and it has limited query capabilities.

Several protocol extensions have been made to the 0.4 version in an attempt to solve these limitations. Improvements such as using “ultrapeers” (super-peers), XML metadata, and parallel downloading are being built in the 0.6 version which is about to be finalized but is already the officially recommended version. This is the protocol used by LimeWire clients.

In 2002, Michael Stokes announced the Gnutella2 [57] protocol which, although inspired by the original protocol and still using the 0.6 handshake mechanism (an attempt to obtain backward compatibility), is more of a redesign than an upgrade of previous versions. A major difference consists in categorizing nodes as hubs (super-peers) and leafs. Hubs may have hundreds of connections and maintain an index of files in its connected leafs. Other new features include an extensible binary packet format, SHA-1 integrity checking, package compression, and a metadata system for file description. Most of old Gnutella clients do not support the Gnutella2 network.

#### *2.3.5.2 BitTorrent*

BitTorrent [58][59] is a P2P file-sharing and content distribution protocol. Files being distributed are described in a metadata document (torrents) as a number of identically-sized pieces, along with the “tracker” info – the peer who maintains a list of nodes participating in the torrent. Clients of the protocol can also implement a trackerless system by using a distributed hash table.

To start downloading a file, peers retrieve the participating nodes list from the tracker in the torrent, and make several requests over distinct TCP sockets to retrieve as many pieces of the file as possible. Although BitTorrent can enhance performance and improve scalability of resource publishers, it provides no

indexing mechanisms. Torrents are usually listed on websites, which provide the searching mechanisms.

#### *2.3.5.3 Napster*

Napster [60] is a known file-sharing program based on a hybrid P2P topology. Every time a node connects itself to the network, it uploads an index of local (shared) files to a central server. All queries made in the network are directed to this server, which looks up in its merged index. While this solution is simple and solves the Gnutella search limitation, it is not however a scalable solution: if there is a traffic peak, the server(s) may be overloaded with requests.

#### *2.3.5.4 FastTrack*

FastTrack [61] is perhaps the most popular P2P protocol, which is used in clients such as KaZaA and iMesh. While based on the Gnutella protocol and also used for file-sharing, it presents some improvements worth noting:

- Automatic super-peer creation: the “best” nodes on the network (processing power, hard disk space, and bandwidth) become super-peers, thus providing (temporary) indexing services for “weaker” nodes. This allows for greater system scalability.
- The file transfer protocol is still HTTP; however FastTrack has algorithms which allow the download from simultaneous sources. It can also resume canceled or interrupted downloads.

FastTrack is a closed proprietary protocol, and for this reason some implementation details are not disclosed.

#### *2.3.5.5 Farsite*

Farsite – Federated, Available, and Reliable Storage for an Incompletely Trusted Environment [62][63] – is a distributed file system which does not rely on a central server. The system logically aggregates several file systems as a single virtual disk. Each network node supplies a local disk quota which can be used by the remaining users.

Farsite allows for such collaboration, without the assumption of complete trust between the nodes, by implementing cryptographic and fault tolerance mechanisms. It also performs the automatic file replication by several peers, which functions as an efficient backup system.

### *2.3.5.6 P-Grid*

P-Grid [64] is a self-organizable P2P system, based on a virtual tree structure. To each peer is assigned part of the tree, and its position is determined by the corresponding binary path. For instance, peer 4 in Figure 2.7 (from the referenced publication) has the binary path 10, which makes it responsible for storing resources whose binary key starts with 10. Redundancy, error resilience, and load balancing can be achieved by placing 2 or more peers in the same path (1-6 and 3-4 in the figure).

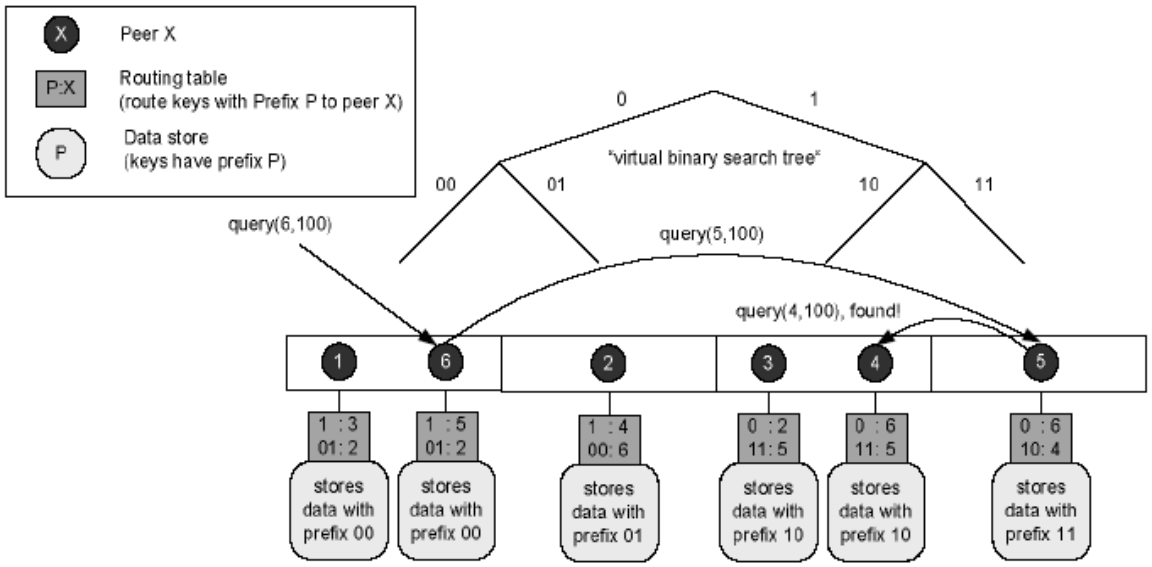


**Figure 2.7 - A simple P-Grid**

Search queries include the desired resource key and nodes retransmit each query to the path which approximates the key, until the final peer is reached. The

advantage of this approach over Gnutella's is that the tree structure makes the query route to be oriented, which allows a reduction in the network traffic.

Unlike other structured P2P systems, the peer identifiers are independent of the paths identifiers, which are dynamically changed by the maintenance protocol.

### *2.3.6 P2P-based digital libraries*

While most of the available P2P protocols aim the sharing of files, there are however other application scopes. In this section, we review the most important projects and frameworks in the scope of digital libraries.

#### *2.3.6.1 P2P-4-DL*

P2P-4-DL [65] aims to build a system for digital libraries which operates in a P2P network. It uses a brokered approach, by storing in a single node the global resource index. There is no replication or load balancing mechanism, as documents always remain only in the owner node.

#### *2.3.6.2 Edutella*

Edutella [66] is a P2P network infrastructure based on RDF aimed at the exchange of educational resources (metadata) between academic institutions. It is built on the JXTA framework (see 2.3.7.1) and implements three different services: Query, which uses a query exchange language; Replication, to achieve metadata persistence and availability; and Mapping, Mediation, and Clustering, which perform mapping between schemas, mediate access between services and set up semantic routing and clustering. Edutella does not handle the data itself and is only responsible for the metadata.

#### *2.3.6.3 P2P Digital Library*

P2P DL [67], currently a prototype, is based on the JXTA framework and is a joint work of the University of Edinburgh (UK), the University of Athens (Greece), and the Foundation for Research and Technology (Greece).

In the proposed architecture, nodes should store data organized in RDF schemas. To allow each peer to have its own RDF schema, the P2P DL has a

mapping mechanism which reformulates queries, prior to its propagation, in order to match the information at the remote nodes.

#### *2.3.6.4 FreeLib*

FreeLib [68] is a project from the Old Dominion University Digital Library Research Group, which applies pure P2P techniques in the context of digital libraries.

FreeLib proposes a different approach from other P2P based digital libraries since it is built on top of OAI mechanisms: each FreeLib node is both an OAI service provider (harvests data and provides end user services such as indexing and searching) and OAI data provider (holds and archive of resources).

#### *2.3.6.5 dLibra*

dLibra [69] is a digital library framework developed in the Poznan Supercomputing and Network Center which aims to facilitate the main phases of the digital publication process.

Content management is accomplished by using a hierarchical directory structure. Document versioning is also supported by the framework. A particular version is made public by creating an edition – a set of publication's objects.

dLibra digital library is implemented as a client-server system. In the server side there are a number of modules connected via network interfaces (implemented using Java Remote Method Invocation)

### *2.3.7 Frameworks and platforms*

#### *2.3.7.1 JXTA*

JXTA [70] is an open-source project which consists in a group of open and generic protocols to connect heterogeneous devices in a P2P network. The Java based framework aims the creation of an interoperable and platform independent P2P network.

Although JXTA represents data in XML, its protocols are not based on standards.

Its architecture (Figure 2.8, from the JXTA documentation) is composed by three layers:

- *Core*, which supports services and applications built with JXTA, defines mechanisms for managing, publishing and discovering groups (*Peer groups*), the communication methodology (*Peer pipes*), and controlling, prioritizing, and monitoring access (*Peer monitoring*);
- *Services*, in which access libraries are made available to the upper layers; some indexing, searching, and sharing services are implemented;
- *Applications*, the upper layer in JXTA, uses the functionality provided by *Core* and *Services* to create specific applications.

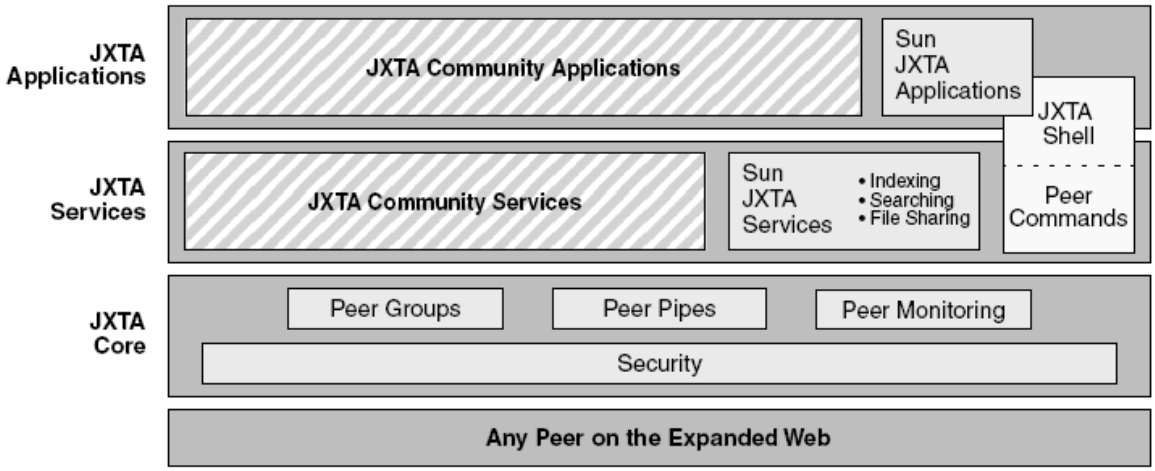


**Figure 2.8 – JXTA architecture**

JXTA defines the following protocols:

- *Peer Discovery*, used to find nodes, groups or other advertised resources;
- *Peer Resolver*, generic set of queries for finding information;
- *Peer Information*, used to determine other nodes capabilities;

- *Rendezvous*, to propagate messages;
- *Pipe Binding*, which allows peers to advertise resources;
- *Endpoint Routing*, protocol which allows peers to use routers to find connections to other nodes.

Besides Edutella, (section 2.3.6.2), there is a large number of projects in a wide variety of fields associated with the framework. For example, jxta-cad is a community effort to adopt JXTA in Computer Aided Design, and trinytalk aims to develop an instant messaging system tool for wireless users based on voice.

Both the JXTA framework and the JXTA-SOAP project were used in this doctoral work and will be referred to later.

#### *2.3.7.2 Windows Peer-to-Peer Networking*

Shipped with Windows XP SP2 and Windows Vista, the Microsoft Windows Peer-to-Peer Networking component allows to create P2P applications which do not require central servers. The platform has the following characteristics:

- End-to-end connectivity, which uses the IPv6 protocol to assure the connection between nodes without compromising security;
- Peer Name Resolution Protocol (PNRP), a protocol designed to allow scalable and secure name registration and resolution;
- Ability to create and organize peer groups, in which information can be synchronized and isolated from outer nodes.

The intended usage scenarios of the framework include real-time communication, collaboration, content distribution, and distributed processing. Its architecture is depicted in Figure 2.9 (from the website) and is divided in the following modules:

- *Graphing*, responsible for maintaining a set of connected nodes (graph) and providing flooding and replication of data;

- *Grouping*, which is the security layer provided by default on top of a graph – it defines the security model behind group creation, invitation, and connection to the group;
- *Name Service Provider (NSP)*, which provides a mechanism to access an arbitrary name service provider (the PNRP in Windows P2P Networking);
- *PNRP*, for P2P name resolution.
- Identity Manager, which enables the creation and management of P2P identities.

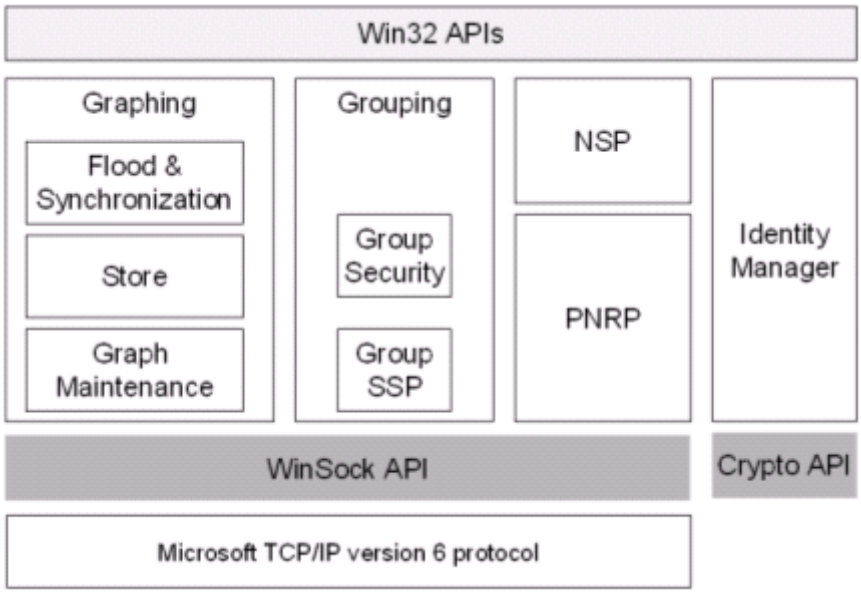


**Figure 2.9 - Windows P2P Networking architecture**

### *2.3.7.3 CSpace*

CSpace aim is to provide a platform for secure, decentralized, user-to-user communication. It is developed in Python, uses OpenSSL for cryptography, and a distributed hash table (DHT) based on the Kadmelia protocol, where a mapping between the user's public key and his IP address is created. User identity is accomplished using 2048-bit RSA keys.

At the time of writing CSpace was still in beta status and the available applications were limited to text chat, file transfer, and remote desktop based on the Virtual Network Computing (VNC) platform-independent system.

### 2.3.8 Other applications

P2P networks can also benefit a wide range of social and entertainment applications, such as instant messaging [71], web television/P2PTV [72], social networking [73], and gaming networks [74][75], especially in massively multiplayer games.

In any case, the goal behind the use of P2P is to use shared resources to increase performance and lower the costs inherent from high bandwidth centralized services.

## 2.4 GRID

In 2001 a generic architecture for Grid systems was proposed [76], which became the reference for many current implementations, such as Globus.

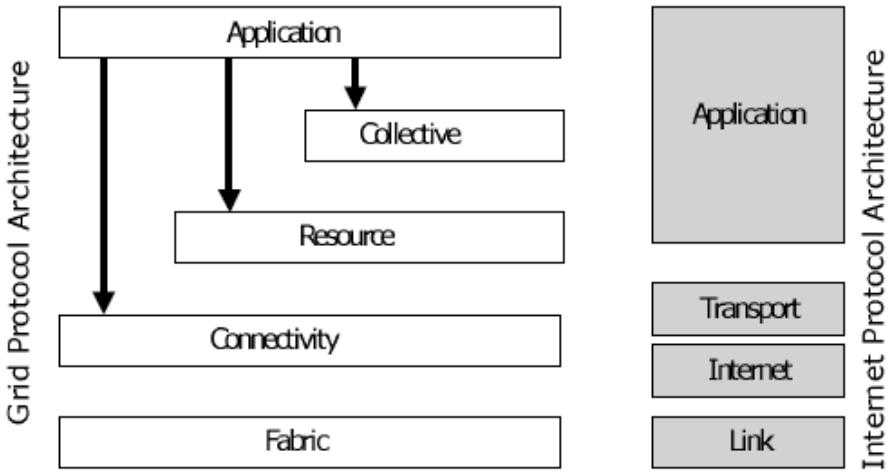


**Figure 2.10 - The Grid vs. the Internet protocol architectures**

The architecture depicted in Figure 2.10 (from the referenced publication) presents an abstract structure composed by a small number of fundamental blocks:

- **Fabric –** This layer provides the system resources (catalogs, memory, processing cycles, etc.) according to an access protocol; depending on the underlying hardware, each resource implements specific operations and has a description mechanism which allows discovering the structure, state and capabilities of resources.
- **Connectivity –** The Connectivity layer defines the communication and authentication protocols required in Grid specific network transactions; these protocols allow the sharing of resources in the *Fabric* layer and provides cryptography and authentication mechanisms; according to the architecture specification, implemented authentication solutions should have some characteristics such as single sign-on, delegation, and integration with local/custom security mechanisms.
- **Resource –** This layer defines protocols to securely negotiate, initialize, monitor, and control individual resources; two protocol classes are defined – Information (used to obtain information about configurations, state, restrictions, etc.) and Management (used to manage the access to shared resources).
- **Collective –** In this block resides the responsibility of coordinating multiple resources; unlike *Resource*, this layer defines protocols associated not to a single resource but instead to the interactions between collections of resources.
- **Applications –** Finally, the *Applications* layer is composed by the applications which operate on top of a given VO. Figure 2.11 (from the referenced publication) depicts the proposed architecture in more detail.

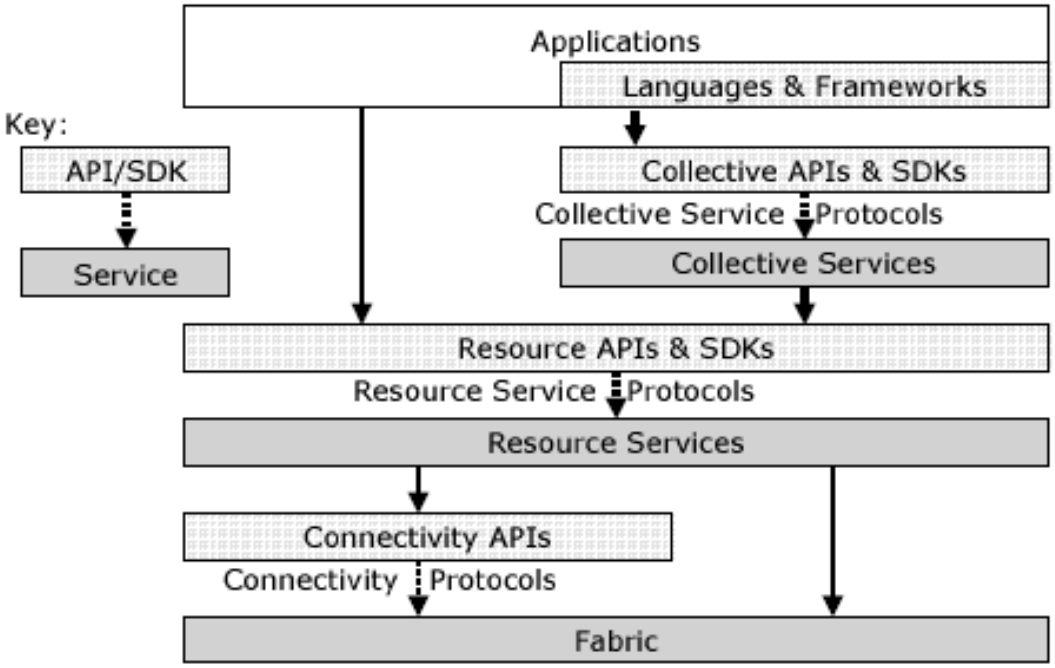


**Figure 2.11 - The Grid architecture**

### *2.4.1 Globus*

The Globus Alliance [77] researches and develops Grid technologies and the Globus Toolkit is the main result of such research. This open source software was one of the first large-scale implementations of the OGSA specifications, and includes several components to monitor, discover, and manage resources.

In the last years, the Alliance has made an approach towards Web Services, using an OGSA compliant architecture (Open Grid Services Architecture) [78] in order to create a distributed platform based in the OGSI infrastructure – the Open Grid Services Infrastructure [79]. The adoption of these concepts lead to the creation of the Grid Services notion, which allow the integration of distributed, heterogeneous and dynamic resources and systems, by defining standard interfaces and behaviors.

Defined as part of the Globus Toolkit is GridFTP, a standard file transfer protocol for use with Grid computing. Its goal is to provide a high-performance,

secure, and reliable transfer protocol based on the regular FTP protocol. It has been the single standard to be widely accepted by the Grid community [80].

### *2.4.2 GridIR (or GIR)*

GridIR [81] is a distributed architecture designed for information retrieval. This retrieval is implemented by using Grid computing tools, and creating a common infrastructure for distributed information systems.

The main characteristics of GridIR are:

- The ability to perform distributed searches;
- The creation of standard based methodologies to distribute the aggregation, processing, and indexing of resources;
- It allows to dynamically create information retrieval systems;
- It allows to create and customize security models specific to each VO;

The GridIR architecture is based on the implementation of three autonomous and distributed services: *Collection Manager* (which monitors catalog documents and issues re-indexing requests), *Indexing/Searching* (indexes the repository documents and creates searchable data bases) and *Query Processing* (provides single access point for multiple indexing services and performs pre- and post-processing of queries and results).

It is worth noting that each search result is simply an URL, which can then be retrieved using an Internet protocol. Each of these services can be dynamically created to serve a VO or connect several VOs. Figure 2.12 represents the simplified architecture.

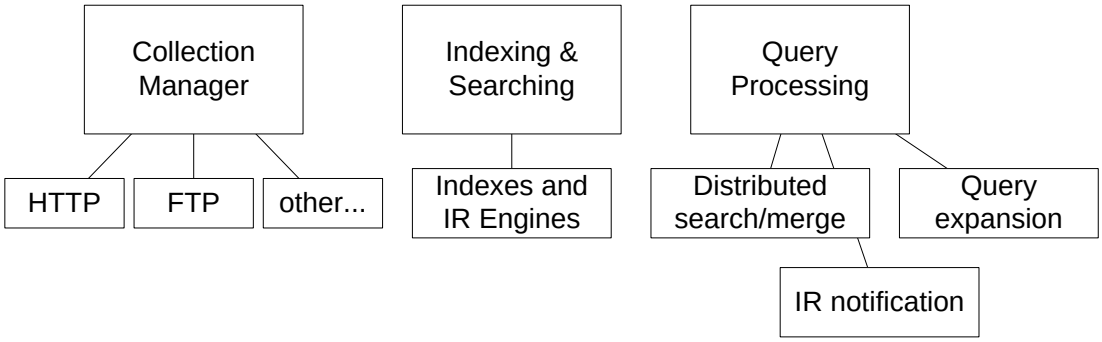


**Figure 2.12 - GridIR architecture**

### *2.4.3 Alchemi*

Most current Grid applications were developed for UNIX-like operating systems, which reduces its applicability for Windows users. To circumvent this issue, the Alchemi [82] framework was created, which is implemented using the Microsoft's .NET platform. Alchemi's main features are:

- Aggregation of computers without a centralized file-sharing system;
- Hierarchical organization and cooperation of Grids;
- Object-oriented programming model;
- Web Service interfaces to allow the interoperability between heterogeneous platforms;

A scenario where a modular architecture uses Alchemi and other Grid technologies (such as Globus Toolkit) is depicted in Figure 2.13.

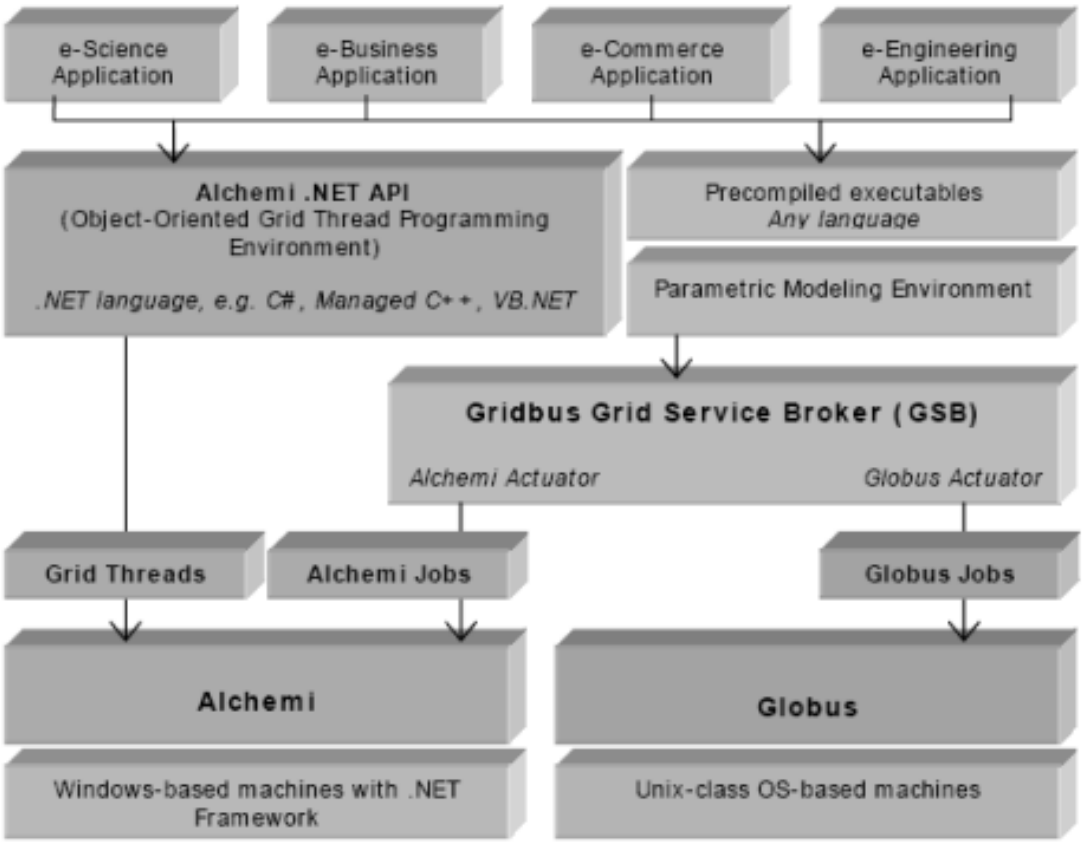


**Figure 2.13 - Integrating Windows and Unix-like resources**

## 2.5 SERVICES ORIENTED COMPUTING

Service oriented computing has been a popular research topic in the last years. The basic principle behind service orientation is that distributed, modular, autonomous and interoperable services available in the network can be (re-)used to enhance or extend application capabilities or even to perform some of its core functionalities. It has become one of the main drivers for the software industry [83].

Several concepts based on service orientation have surfaced in recent years. Some of the most popular are:

- Service-oriented architectures (SOA) – an infrastructure in which business processes are implemented through distributed services (typically Web Services) [24][84];

- Software as a service (SaaS) – a model of software licensing in which services are provided on demand [85].
- Cloud computing – the availability of services and resources on the internet, which can be consumed (and meshed) in a variety of applications. Unlike the previous concepts, cloud computing is commonly thought as collections of services which can also be consumed for personal use (such as in blogs) [86].

Properly managing and consuming a wide range of available services presents a problem of standardization of those services. Even in the case where all services are SOAP Web Services, a standard and widely adopted technology, it is required to define a priori which methods, data structures and interactions will be used.

In the simplest case, consumers may use only a few services separately to add extra functionality or perform very specific tasks, and in this case developers can easily perform a service call or create a service proxy. However, service orientation advantages are only being partially explored in this scenario.

Service orientation allows creating complex, composite services which are a logical aggregation of other services in a flow – the business process. Orchestration and choreography languages allow defining information flows and creating these composite services to accomplish processes.

A combination of SOA, business process choreography and Web Services can bring numerous advantages for businesses [87]:

- Higher automation and process integration;
- Increased productivity with cost reduction and better performance in process execution;
- Simplification in the reuse of services and components;
- Standardization allows replacing unsupported components by commercially available products.

### *2.5.1 Core technology*

SOA refers to a new architectural style which is not tied to a specific technology. At most, common SOA frameworks and platforms generally use XML enabled services. SOA can be implemented using a wide range of technologies, from which RPC, SOAP, Web Services, and REST are the most popular.

#### *2.5.1.1 Web Services*

Although service-oriented architectures are not bound to a specific technology or protocol, Web Services [88][21] became the standard for its implementation. Web Services, an extensively XML based standard, use the SOAP protocol for the invocation of services and WSDL for describing the interfaces. The following XML is the WSDL description for a Web Service with a single method (Add) which adds two integers.

```
<?xml version="1.0" encoding="utf-8"?>

<wsdl:definitions
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:s="http://www.w3.org/2001/XMLSchema"
      xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
      xmlns:tns="http://Math" targetNamespace="http://Math"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:types>
    <s:schema elementFormDefault="qualified"
      targetNamespace="http://Math">
      <s:element name="Add">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="a"
               type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="b"
               type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="AddResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="AddResult"
                  type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
```

```
  <wsdl:message name="AddSoapIn">
    <wsdl:part name="parameters" element="tns:Add" />
  </wsdl:message>
  <wsdl:message name="AddSoapOut">
    <wsdl:part name="parameters" element="tns:AddResponse" />
  </wsdl:message>

  <wsdl:portType name="wsMathSoap">
    <wsdl:operation name="Add">
      <wsdl:input message="tns:AddSoapIn" />
      <wsdl:output message="tns:AddSoapOut" />
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="wsMathSoap" type="tns:wsMathSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Add">
      <soap:operation soapAction="http://Math/Add" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="wsMath">
    <wsdl:port name="wsMathSoap" binding="tns:wsMathSoap">
      <soap:address location="http://localhost/wsMath " />
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

Below the declaration of all namespaces used in the document, the WSDL defines the types of messages (Add and AddResponse) and its variables (a, b, and AddResult). The interfaces (the wsMathSoap portType) are then defined. The binding defines the operations for the interface and associates it to a specific transport protocol (HTTP). Finally, a service (wsMath) is declared as a binding at a specific location (http://localhost/wsMath).

Several other specifications were created in the Web Service universe to aid the completing of certain tasks: Universal Description Discovery and Integration (UDDI) and the WS-Discovery specification are used for service discovery, WS-Routing is a protocol for asynchronous message routing over several transports,

the WS-Eventing and WS-Notification specifications allow the subscription to events and notification messages, and many more.

The main advantage of using Web Services, which are driven by software giants such as Microsoft and IBM, is that it provides an interoperable and autonomous mechanism for describing and invoking remote services. Services and data provided by a Java Web Service can be consumed in same way by clients written in .NET and running in Windows or any other platform or operating system. Some criticism regarding Web Services is often related either to its complexity or due to performance concerns, since it is uses XML, SOAP, and HTTP.

#### *2.5.1.2 SOAP*

Once the acronym for “Simple Object Access Protocol” (definition abandoned in version 1.2), SOAP is a protocol for the exchange of XML based messages over the network, and uses the Internet application layer protocol (either HTTP or SMTP) as a transport protocol.

SOAP is platform and language independent, extensible, and based on widely adopted standards. It does, however, rely on a rather verbose XML format which can degrade performance (parsing time, network bandwidth).

#### *2.5.1.3 RPC*

RPC or Remote Procedure Call is a generic technique which allows a program running on a computer to call (invoke) procedures provided in a different computer [89].

A set of tools are responsible for making the communication details transparent to the developer; however, extra care is usually needed to catch and process unexpected network problems.

The history of the RPC concept dates back at least three decades and there are several models and implementations. The first popular implementations were

Xerox's Courier and Sun's UNIX RPC. Currently, every major software vendor has its own solution, such as Java RMI or Microsoft .NET Remoting.

The main problem with RPC is that there many different protocols and technologies to implement it, which are commonly incompatible between each other.

#### *2.5.1.4 REST*

REST or Representational State Transfer [90] is a style of software architecture based on the concept of resources which are addressed by identifiers such as a URIs. The acronym was first coined by Roy Fielding in its doctoral dissertation [91].

The motivation behind REST was to capture the characteristics which made the Web simple and successful – REST reflects the architectural style of Web itself. The most common REST application is based on the HTTP protocol and the GET, POST, PUT, and DELETE verbs. For instance, a school's web site may provide a list of students at the URL:

http://myschool.com/students

And the "representation" of the student with ID 238709 could be available at:

http://myschool.com/students/238709

For other applications to communicate with this system there must be a sequence of actions very similar to those triggered by a user's browser. Both the list of students and the data of a particular one could be retrieved by issuing a GET verb on the students URLs, very much like the HTTP headers the browser would send to the server. Creating or updating student files would require issuing POST or PUT headers, while deleting them would require a DELETE verb.

Despite its apparent simplicity, REST does place some implementation issues. While Web Services provide standard mechanisms to describe service interfaces and data/message types, REST does not – it is simply an architectural

style and does not explicitly define the service model it exposes. Also, while simple and atomic read actions could be easier to implement with REST, more complex operations (such as transactions) may be simplified using SOAP tools.

### *2.5.2 Service orchestration*

In service oriented architectures there are several distributed services which can be used by an application to perform tasks of distinct complexity. Even in the case where all services are SOAP Web Services, a standard and widely adopted technology, it is required to define a priori which methods, data structures and interactions will be used. Orchestration and choreography languages allow defining information flows and creating composite services to accomplish processes.

#### *2.5.2.1 Orchestration vs. choreography*

Although both service orchestration and service choreography serve the same purpose – to achieve a certain goal based on the cooperation of several intervenients – they relate to two distinct concepts. The main difference relies at the level of control:

- in an orchestration there is a “maestro”, some participant who controls and instructs the process interpreters;
- in a service choreography all interpreters know and execute their role without external control.

At the description level, orchestration is focused on the behavior of an intervenient, which executes a certain task of the process. The process definition is bottom-up: it starts with the declaration of individual tasks followed by the definition of the collaboration.

Choreography defines global, peer-to-peer, and interoperable collaborations. Its intervenients act as stateful nodes in a coordinate fashion and there is not a centralized management peer. Process definition is top-down: from the global process to its tasks.

#### *2.5.2.2 BPEL*

Both IBM and Microsoft had proprietary languages for service orchestration – WSFL and XLANG, respectively – but ultimately decided to merge the specifications into the new BPEL4WS (Business Process Execution Language for Web Services) language, later renamed into WS-BPEL or simply BPEL. The BPEL language allows defining composite services through a logic control flow of existing Web Services. It supports synchronous and asynchronous interactions, flux control and transaction compensation (instead of rollback mechanisms).

BPEL makes use of several XML standards: WSDL 1.1 and XML Schema 1.0 (data model), and XPath 1.0 and XSLT 1.0 (data manipulation).

The initial goals of BPEL were [92]:

- To define business process that can interact with external entities through XML and Web Services, and are themselves expressed as Web Services;
- To use Web Services in a modular and composable fashion;
- Do not define any design methodology or graphical representation for processes;
- Define a set orchestration concepts to be used by both the external (abstract) and internal (execution) views of a business process;
- Provide simple data manipulation functions needed to define process relevant data and control flow;
- Support an identification mechanism for process instances;
- Support the implicit creation and termination of processes;
- Define a long-running transaction model based on compensation and scoping to support failure recovery;

The following activities are defined in the BPEL 2.0 standard [93]:

- Basic activities:
    - Invoke: invoke a method from a service provider
    - Receive: wait for external invocation of a method
    - Reply: send a response to a previously received request

  - Assign: update (and/or copy) variables values
  - Throw: signal an internal exception
  - Rethrow: re-send the exception signal
  - Wait: standby with no activity (sleep) for a period of time
  - Empty: define an null activity (sometimes required for synchronism)
  - Exit: terminates the process instance
  - ExtensionActivity: not defined
- Structured activities:
  - Sequence: execute activities in sequence
  - If/ElseIf/Else: execute activities if conditions met
  - While: execute activities while condition met
  - RepeatUntil: execute activities until condition met
  - Pick: execute one of the activities according to an event
  - ForEach: loop activities execution a defined number of times
  - Flow: encapsulate activities to be executed in parallel

#### *2.5.2.3 Engines and tools*

There are currently several orchestration engines available, both commercial and open source. Every major software vendor has its own BPEL product, which reflects the importance given to the topic:

- ActiveBPEL [94]: a comprehensive BPEL open source IDE developed in Java (commercial products also available);
- ODE [95]: the Apache family engine, which evolved from the discontinued Agila BPEL;
- WebSphere Process Server [96]: the IBM process engine executes in the WebSphere Application Server Java EE platform;
- BizTalk Server [97]: Microsoft's process server (previously based in XLANG) allows transforming BPEL orchestrations into BizTalk descriptions and vice versa;
- Oracle BPEL Process Manager [98]: the engine previously known as Collaxa BPEL Orchestration Server, later acquired by Oracle, executes as a

J2EE application on Oracle Application Server, Jboss, BEA Weblogic e IBM WebSphere;

- Netbeans SOA [99]: the IDE's SOA pack integrates a BPEL project type;
- Eclipse BPEL [100]: a BPEL plug-in (designer and runtime included) for the popular open-source IDE.

#### *2.5.2.4 Other languages*

The Web Service Choreography Description Language (WS-CDL) [101] is a XML based language which allows specifying peer-to-peer protocols in which there is no central control and every peer remains autonomous. WS-CDL abstracts itself from the type of processes involved; unlike BPEL it is not based on WSDL, although it can be used with Web Services. Rather than being involved in the execution or implementation of processes, it defines a controlled and complementary behavior by each party (i.e. the interactions between services), which can be implemented using different technologies. WS-CDL is not as widely accepted (and supported) as BPEL.

XML Process Definition Language (XPDL) [102] is a language standardized by the Workflow Management Coalition to design processes. The 2.0 version contemplates the use of extensions to allow representing all aspects of the Business Process Management Notation (BPMN). While BPEL defines an orchestration, the interactions and data flows, XPDL is responsible for the storage and interaction of process diagrams, although it is primarily associated with traditional workflows [103]. Hence, two engines can share the same XPDL definition e use distinct execution mechanisms (Figure 2.14, from Swenson's website).

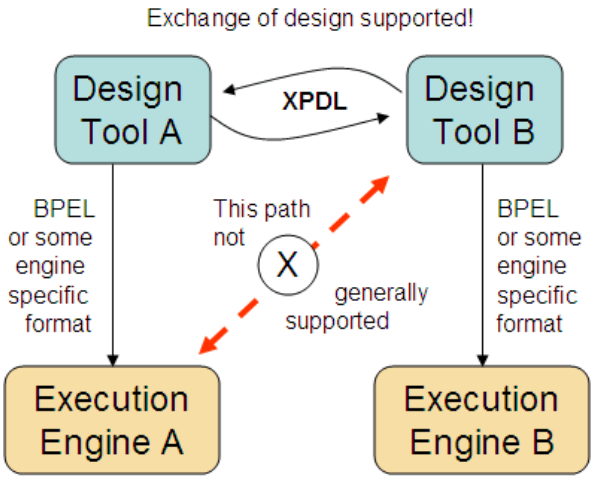


**Figure 2.14 - The exchange of process designs**

The BPML language was recommended by the Business Process Management Initiative (BPMI), currently abandoned in favor of BPEL4WS.

A few other works have tried to accommodate a wider range of services into BPEL processes.

In [104] a platform is presented for the hybrid composition of both Web Services and Grid Services. BPEL only supports Web Services, a limitation the authors circumvent by creating the concept of Virtual Web Services, which encapsulate Grid Services. The OWL-S [105] ontology is also used to achieve a richer description for the Web Services.

In light of the recent popularity of REST, some work has also been made in order to allow RESTful services to be supported by BPEL engines. The professional edition of ActiveBPEL, for instance, supports activities that handle messages based on the REST architecture rather than WSDL operations [106]. Some authors [107] have also proposed to natively support the composition of RESTful services with business processes using BPEL extensions.

#### *2.5.2.5 Decentralization*

BPEL engines (as all major engines for orchestration in proprietary protocols and schemas) are installed on a server and are responsible for interpreting the orchestration description, invoking services, monitoring data flow, storing state for long-running transactions and eventually aggregating results.

Although BPEL consists in the execution of distributed Web Services, its orchestration is in fact centralized. In a data-intensive process, the communication of inputs and outputs between the services and the “maestro” (the service orchestrator) can become very inefficient.

Orchestrations described with the BPEL specification have several limitations which make them less than ideal to be used in a dynamic scenario such as a P2P network. A natural limitation to BPEL consists in the lack of support for dynamically discovering and assigning service providers. A process description must be completely defined with its providers from the beginning. On the other hand, the specification defines activity execution in a sequential manner and there is no event based model [87].

Distributing the orchestration process by the service providers has several advantages, especially in high load scenarios and/or when there is a high amount of data to be transferred between services. A careful partitioning process can reduce the number of messages and amount of data transferred and increase throughput.

There are, however, a number of issues which make distributing tasks a non-trivial procedure:

- Scenarios in which parallel operation is important, and where there are complex inter-service dependencies, can be difficult to distribute;
- In a centralized engine it is easy to determine the current process state and where the execution is at each instant, while on a decentralized orchestration there may need to exist feedback mechanisms to the machine which initiated the process;

- Delegating service orchestration and invocation in an untrusted network can be unattractive.

To circumvent this possible bottleneck and try to boost performance in the execution of complex processes, some solutions have been proposed.

One possible technique proposed by IBM researchers [108] consists in partitioning a BPEL instruction sequence into a set of distributed processes, eventually reordered but with the same final output, under the assumption that every node has BPEL runtime capabilities. The algorithm consists in dividing BPEL activities into fixed (receive, reply, and invoke) and portable (other) ones. Each fixed activity is aggregated with a process services (receive/reply pair with the entry point), while portable ones can be moved. The final arrangement consists in partitions with one fixed activity and zero or more portable ones. According to the authors, partitioning processes using this algorithm may increase its throughput 30% at normal system load and by a factor of two under high load. Figure 2.15 (from the article) depicts an example of a composite service executed with centralized and decentralized orchestration.

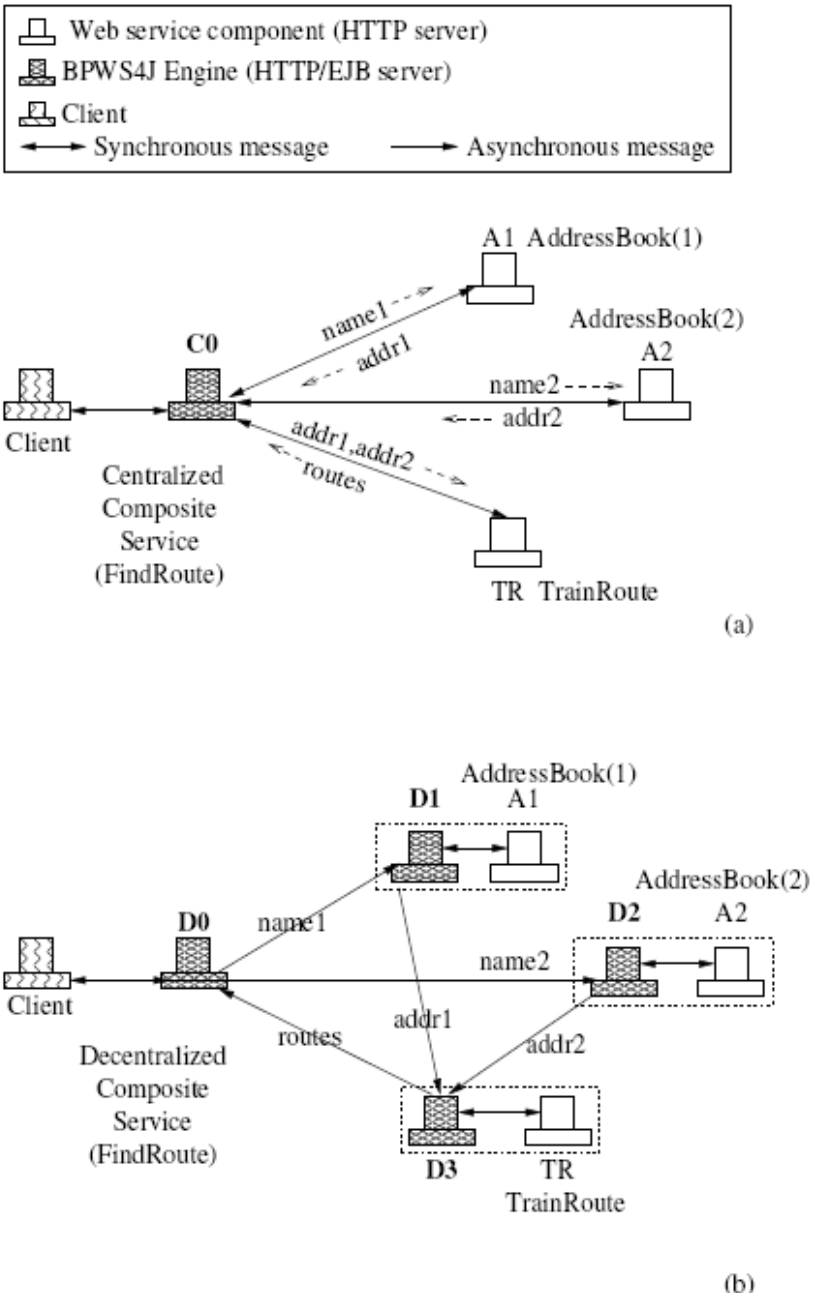


**Figure 2.15 – Centralized (a) and decentralized (b) orchestration**

Another proposal [109] consists in decentralizing the flow control and dynamically selecting roles. The presented approach considers only simple flows, without synchronization, restrictions, or error handling. It is adopted a stateless model: a node, after executing an activity, transfers all state information to the next node.

Other authors [110] propose enacting decentralized workflows with a different approach. It is based in basically two steps: 1) process segmentation (manual, for now) is made by analyzing the physical infrastructure and annotating the process with information about how activities and variables are mapped into the participants, and 2) transformation of a BPEL process into a decentralized model called Executable Workflow Networks (EWFN).

Khalaf et. al [111] discuss how to maintain data dependencies when partitioning a BPEL process into fragments. The proposal aims to tackle issues that arise from parallelism and shared variables. Our work is for now focused on the technology integration for simpler processes.

## 2.6 RESULTS

We have started this chapter by analyzing existing digital library management systems and related technologies. We then studied in detail P2P data structures and topologies. As discussed, a hybrid topology seems to better fit the needs of a digital library. However, such analyses are usually made on large networks where high latency and communication costs play an important role. Let one however consider the case of a small digital library, whose services are provided by a small number of machines connected by a high-speed LAN network. There is probably a high percentage of small to medium universities and organizations with valuable scientific and historical repository facing an identical scenario. In order to evaluate if the benefits of having super-peers in such environments would still be so apparent, we conducted an experiment [37].

The benchmark was made on a 100 Mbit LAN network where 7 peers with Lucene indexes were connected. The test was divided into two scenarios: in the first stage, a peer was designated as a super-peer and hence search queries were centralized; on a second stage, peers were set to work on a completely decentralized topology. In both scenarios a series of tests were made by performing search queries which would return from zero to about 8.000 results. Figure 2.16 depicts the test results in both stages.


**Figure 2.16 – Search performance with and without a super-peer in a small LAN network**

Results show that even in a small and fast network a super-peer allows for better search performance than that offered by a decentralized topology. However, the difference tends to be marginal except when the number of results is very small, a case in which the overhead involved in establishing connections with other peers becomes important. It also indicates that returning a very high number of search results may lead to undesirable response times, even in small and fast networks. In order to keep the search performance acceptable, the number of hits should be limited so that results arrive in less than a second. Alternatively, they should be presented to users as they are sent by each peer.

## 2.7 SUMMARY

This chapter has overviewed a range of existing concepts, technologies, and standards related to digital libraries or what can be adapted to build or improve such systems.

We started by summarizing the vision of future digital library architectures from workgroups in the DELOS network of excellence. We then reviewed the most popular digital library management systems and discussed its advantages and limitations, taken into account in the development of SInBAD.

In the following sections an overview was made on three of the most important concepts to have appeared recently – P2P networking, Grid computing, and service oriented computing. The most relevant state of the art of these concepts was discussed, particularly in the scope of digital library development.

The limitations found in existent DLMS (poor search granularity, rigid metadata models, reduced Web Service support, etc.) led to the creation of a new digital library system for SInBAD, which had to be a flexible and interoperable system for storing and viewing very heterogeneous resources. To use a service oriented architecture and technologies was therefore a requirement set from the beginning.

Also, and although P2P technologies were not included in the first version of SInBAD, we have discussed the advantages P2P may bring to a DLMS and decided to start designing an alternative architecture based on both SOA and a hybrid P2P network. To validate that hybrid topologies remain a valid choice even in very small LAN networks, we benchmarked the search performance on such a scenario with and without a super-peer and confirmed the assumption.

# CHAPTER 3 – SInBAD

## *3.1 INTRODUCTION*

In the late 2004 the University of Aveiro, funded by the Aveiro Digital 2003-2006 program, started to remodel its internet sites and applications and develop new ones. Most sites installed at the institution by that time provided standalone services, but the paradigm dramatically changed. The new applications to be produced – the library's site, departmental pages, user management services, and many more – had to integrate and interoperate with each other, thus creating a network of cooperative and complementary systems.

In this new scenario, each system is solely responsible for its own data and must provide a predefined set of services and data when another system makes a request. In the scope of this project, the author was an active member of the conception and development of SInBAD [25][26][27] – an integrated system for the digital library and digital archive from the University of Aveiro.

The objective of SInBAD was to design and implement the university's institutional repository, which should have a web application with the purpose of

allowing the storage, cataloging, searching, and dissemination of the digital assets. These assets are heterogeneous contents originating from several departments and services within the campus:

- The university's Library supplies digitized books and journals to support classes, and theses and dissertations from its students and researchers; one of the largest national collections of posters (political, event-related, and others) is managed by the Library;
- The External Relations Service provides the historical archive of photographs of the campus and its events;
- CEMED – the Multimedia and e-Learning Center – is responsible for producing and providing the video archive; "3810", a television program which aired in one of the national televisions, was the main collection;
- The Centre of Jazz Studies – CEJ – digitizes a large collection of audio records (CDs and vinyl albums), and jazz-related books and magazines;
- The museological archive digitizes and catalogs items from its three main collections (Ceramics, Iron, and Glass).

Upon such scenario, the design of the architecture had to take into consideration several issues in its conception:

- The heterogeneity of the resources (books, photographs, audio, videos) makes it difficult to use a common metadata standard that appropriately describes each type of content;
- The multitude of providers demands for a decentralized control mechanism;
- The volume of the data was expected to increase very rapidly, since a large part of the system's items is multimedia.

Furthermore, the goal was that the system should interoperate with other campus applications, both legacy and ones being created at the same time in the scope of the Aveiro Digital programme. Such integration included the new centralized authentication mechanism, but it was mainly related to interoperating with other data repositories.

It therefore became apparent the need for a service-based architecture, which could interoperate with other systems in a standard and controlled way.

## 3.2 OTHER SYSTEMS

At the beginning of the design process, and regarding to scientific and cultural publications, two existing systems provided most of the bibliographic information of the University: e-ABC (Bibliographic Archive for Scientific production) and Aleph.

### 3.2.1 Legacy applications

e-ABC [112] is used by departments in the University of Aveiro to maintain an updated index of the work of researchers and teachers. e-ABC not only stores the bibliographic references of the work developed in the university but also maintains an authority database. In this database, all known authors are stored along with an historic of its affiliations, and on each paper author information is linked with the authority records.

The system is therefore capable of generating annual production reports for a department or institute, maintaining an updated publications list for authors, and show who published with whom (who published with an author, which departments published together, etc.).

e-ABC was remodeled to use the new centralized user management service and to provide Web Services from which other systems can consume information.

The university's library uses Aleph [113], an integrated library system for the management of the bibliographic entries from most of the existing documents available (not only in the library itself, but also in the archive, the multimedia centre, etc.).

This system cannot be considered a digital library, since only bibliographic information (the metadata) is managed – the documents in digital format are not stored. Metadata is stored according to the UNIMARC standard.

### *3.2.2 New services*

With the shift in the architectural paradigm of the campus applications, new systems and services were created to provide common and centralized functionality to all systems.

One of such systems is the Central User Registry (RCU), which provides unique identifiers to individuals. Before the RCU was created, users in the campus (students, teachers, and staff) had several distinct credentials for the many applications and services. For instance, a student would have a credential to access his e-mail account, another to connect to the wireless network, and yet another to make his inscription in the courses at the academic portal. All other specific applications, developed at a particular department or unit, would also require new credentials.

With the development of RCU, each person was assigned with a unique electronic identity (UU), whose credential should be used to enter in all campus-wide applications. This forced existing applications to adapt its authentication mechanisms and change the structure of its databases, and was established as prerequisite to any new application (since SInBAD's first working versions were created at an earlier stage, the authentication and account management had also to be adapted later on).

Finally, the rethinking of the paradigm of the campus applications coincided with a reformulation of the University's corporate image and the redesign of all its web sites, which from that point on were created with an identical layout. In such layout, a large fixed-sized header was mandatory, created by composing four 240x160 background images, on top of which were layered the University logo, the specific site logo (department, unit, or service), and optional contextual text (such as "home" or "contacts"). To centralize the management of the campus web image and automate some processes the Banners service was created. Applications developed for the University could then use banners provided by this service in REST style requests (either made by a JavaScript component or a .NET

component). A final drop down menu was layered on top of the header by each application in a uniform way.

## 3.3 METADATA

Metadata must be uniformly described using standards, which is not a trivial task due to the heterogeneity of resources. In order to adhere to a simple a generic description standard, which could easily be indexed and used on simple search queries, the widely used Dublin Core Metadata Initiative [42] XML Schema was adopted.

The simple Dublin Core metadata schema consists in 15 generic elements described in Table 3-1 (a qualified schema has also become available, with added elements and refinements). A description is made by attributing values to these (repeatable) elements.

While this set of qualifiers may suffice to generically describe any resource, different document types require more fine-grained definition models to better describe – and later search for – objects. For example, one may wish to store information with more details about a book (the ISBN number, the number of pages), a picture (the dimensions), or a video (the segments in which it is divided). Hence, while the Dublin Core schema was used to describe the common attributes of all objects, the XML was adapted to include description values from other standards.

**Table 3-1 – Simple Dublin Core schema elements**

| Element | Definition |
|---|---|
| **Identifier** | An unambiguous identifier within a given context. |
| **Title** | The title of the resource. |
| **Creator** | The author or entity responsible for creating the resource. |
| **Contributor** | An entity with contributions made to the resource. |

| | |
|---|---|
| **Date** | The date or period of time associated with the resource. |
| **Description** | A description of the resource. |
| **Subject** | The topic of the resource, expressed by keywords or classification codes. |
| **Coverage** | The spatial or temporal scope of the resource, or the jurisdiction under which it is relevant or applicable. |
| **Format** | The format or medium of the resource. |
| **Type** | The nature or genre of the resource. |
| **Language** | The language of the resource. |
| **Publisher** | The entity responsible for the publishing or availability of the resource. |
| **Relation** | A related resource. |
| **Source** | A related resource from which the described one is derived. |
| **Rights** | Information about rights associated with the resource (property, intellectual, etc.) |

Some very specific descriptors were used internally to further describe each object, using the **sinbad** prefix in the corresponding XML documents. The most common one (and the only applicable to every object type) is **changed**, containing the date and author of the last cataloguing update. This does not describe the object itself, and is used for internal purposes only. Other sinbad descriptors are mentioned later in the section.

### *3.3.1 Repository structure*

The repository was logically divided into subsystems and catalogs with a structure derived from content ownership (or publishing entity) and type, as depicted in Figure 3.1.

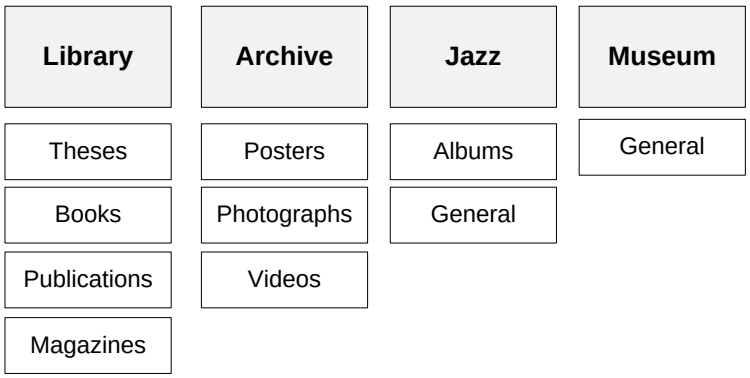


**Figure 3.1 – SInBAD repository structure**

The Library subsystem contains the theses and dissertations, books, scientific publications, and magazines catalogs; the Archive holds the graphic and audiovisual material (except for the jazz content) catalogs; Jazz contains one catalog for the albums and other for jazz-related books and magazines; finally all the museological items are in the Museum subsystem.

To allow for a more fine-grained structure, some catalogs are also organized in collections. The Posters catalog, for instance, is divided into a number of collections, such as Political, Social, Sports, or Concerts, while a photograph may belong to the Academic Ceremonies, Cultural Events, or Scientific Events collections.

Finally, some indexes and vocabularies were created to help better organize the repository, although they do not actually act as containers. For example, a doctoral student is associated with one or more departments in the University (sometimes an external university is also involved), and therefore his dissertation must be associated to these units in a controlled manner.

### *3.3.2 Monographic content*

Due to the limited coverage of the Simple Dublin Core element set, the metadata for monographic objects (books, theses, magazines) is enhanced with descriptors from two other Dublin Core namespaces: the “terms” namespace

(used in the Qualified Dublin Core) and the DC-Library Application Profile (DC-Lib). The metadata schema is depicted in Figure 3.2.

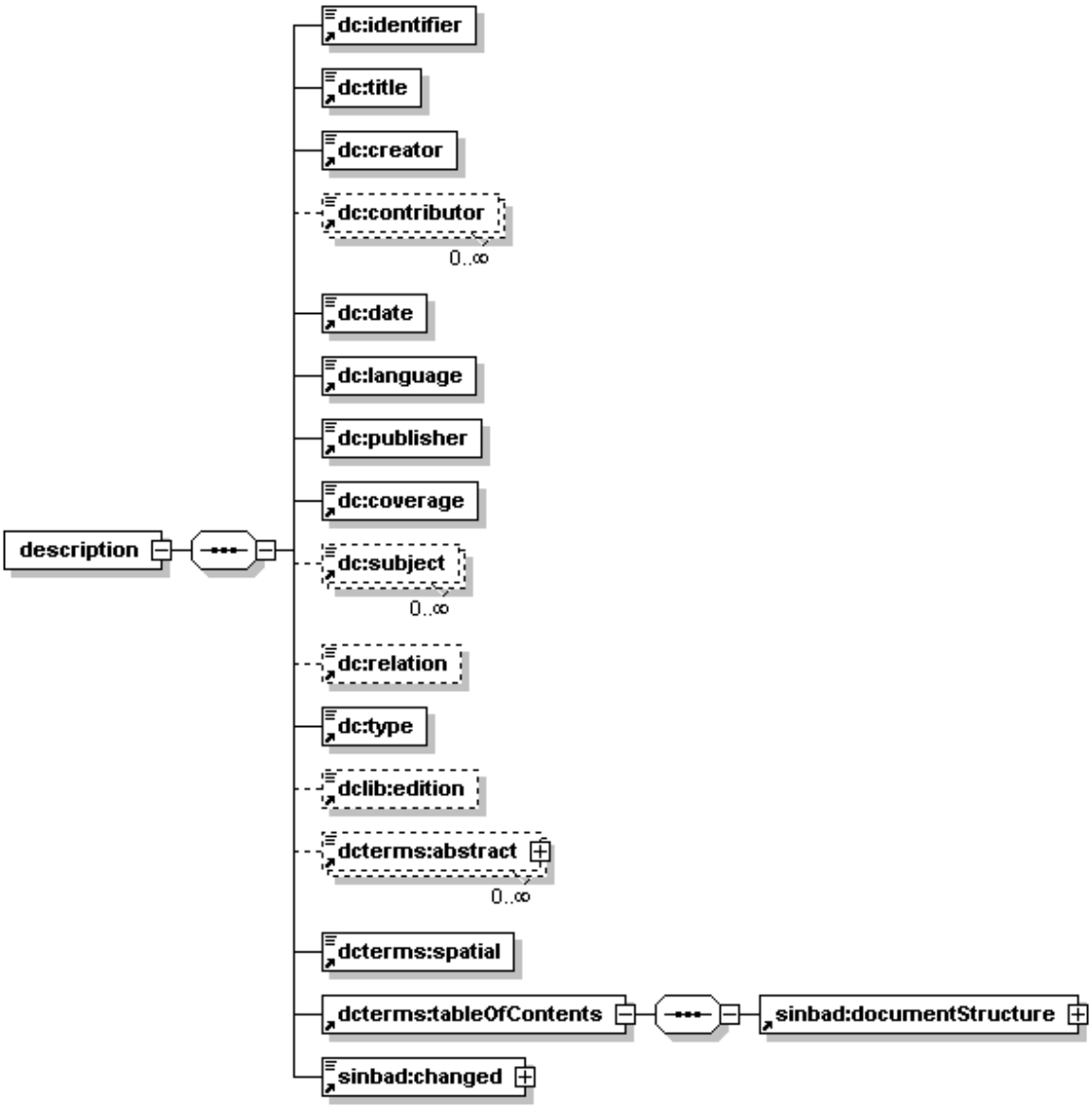


**Figure 3.2 – Metadata[1] for monographic objects in SInBAD**

[1] XML diagrams used in this document use the following notation: a) solid boxes represent mandatory elements, while dashed ones are optional, b) child elements are grouped inside a hexagonal box with 3 dots crossed by a line, c) an hexagonal box with a switch indicates only one of its children should be present, and d) the definition of minimum and maximum quantity is represented numerically below the corresponding element with the **min..max** format.

While most descriptors are self-explanatory, others are worth discussing:

- As with any object in the SInBAD repository, the <dc:identifier/> descriptor holds the unique URL for the object (http://sinbad.ua.pt/[catalog]/[id]);
- For theses, the <dc:coverage/> element is used to indicate the department or unit associated with the object, and the <dc:contributor/> holds the names of the supervisors;
- The description may hold several abstracts, one per language;
- A structured table of contents is stored to allow a better navigation throughout the document later on; a custom <sinbad:documentStructure/> element was created to accommodate it (Figure 3.3).

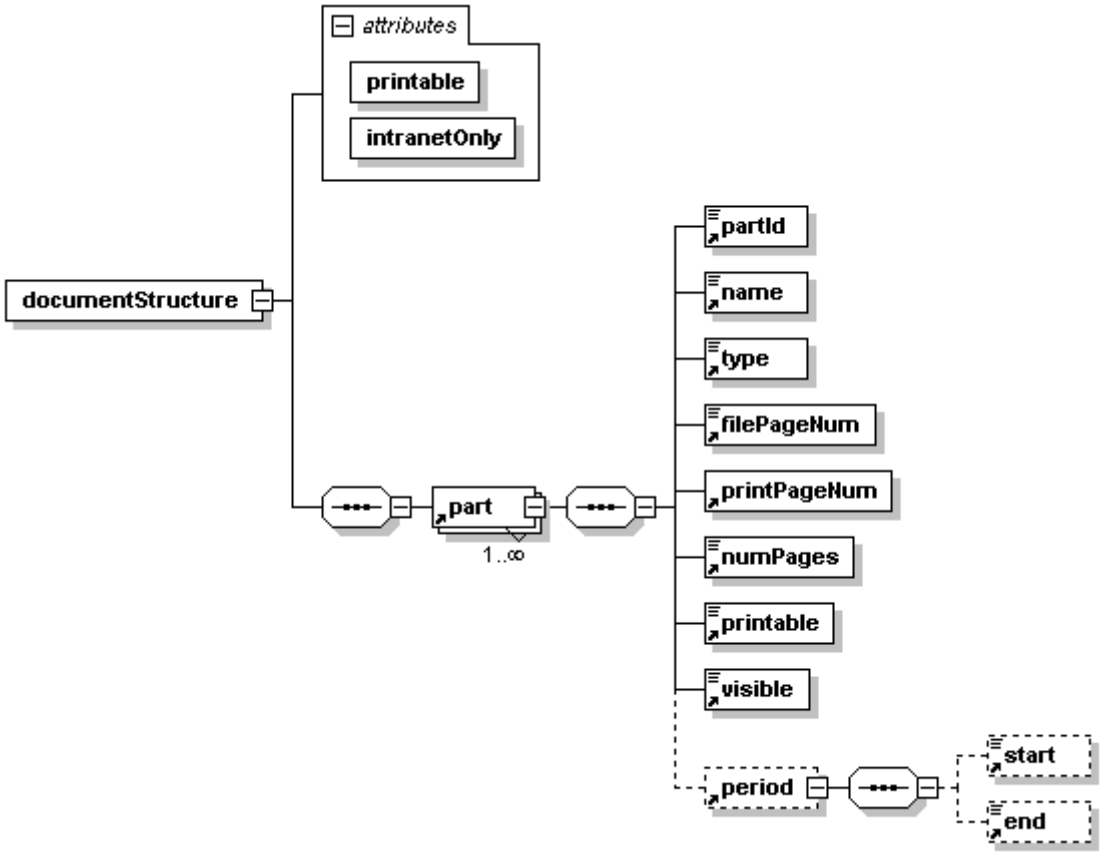


**Figure 3.3 – Structure for table of contents**

The table of contents is composed by a sequence of parts and has two global Boolean attributes: *printable* indicates if users can download or print the document and *IntranetOnly* if the access to outside campus is restricted. Each part (chapter or section) has the following properties: identifier, name, type (either a regular section or an attachment), the first page of section (both physical and textual), and the page count. View and download restrictions, along with an optional period in which they are applicable, can also be applied to each part.

While such model is fit for books and theses, it is inadequate to properly describe scientific magazines and journals. The rationale is simple: while a book or thesis has (generically) document-wide title, authors, and descriptions, a journal contains multiple articles, each with its own title, authors, and abstracts. The magazines catalog has a slight variation in its metadata. While magazines and journals as a whole are described in a similar manner, each part has also its own metadata (Figure 3.4). A reference to the parent object is made through the <dcterms:isPartOf/> element.

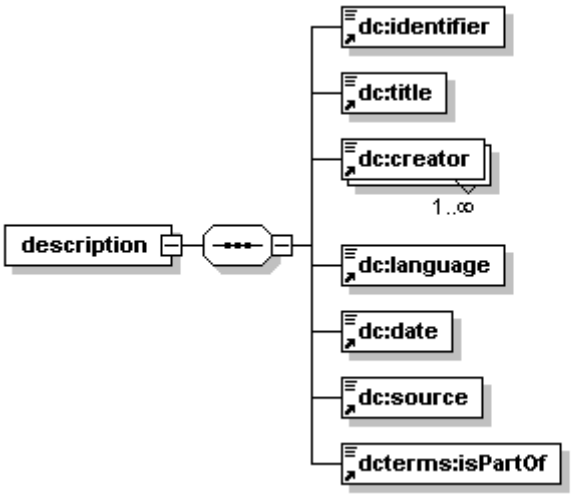


**Figure 3.4 – Metadata for articles in SInBAD**

### *3.3.3 Posters and photographs*

For the image based resources (posters and photographs), the VRA Core 3.0 [114] was used. The VRA Core consists in an element set to describe works of visual culture as well as the images that document them. It therefore offers a number of descriptors that better categorize a graphic object, such as the measurements, the material of which the image is composed, or the style or period. Several of its descriptors match those of Dublin Core (title, creator, date, subject, relation, description, source, rights), making it simple to map between both standards. The metadata for graphical objects in the repository is depicted in Figure 3.5.

Some usage details:

- The <dc:subject/> descriptor is repeatable and used for storing keywords, while the <vracore:subject/> is used to indicate the collection the item belongs to; furthermore, the <vracore:relation.identity/> holds the collection identifier;
- The rights element is used to contain the access rights definition (e.g., a viewLevel of 1 indicates everyone can view the object, while a viewLevel of 2 restrains the access to editors);
- When the metadata is imported from an existing record of the Aleph system, its former identifier must be stored in the <vracore:idNumber.formerRepository/> element.

### *3.3.4 Multimedia*

The generic metadata used for videos in SInBAD is based solely on the Simple Dublin Core terms and on <dcterms:tableOfContents/>. Inside this element, a detailed MPEG-7 [115] based metadata is placed. The complete structure of such XML is too complex to represent in a model, but the main subset is depicted in Figure 3.6.

The most fine-grained description which can be made on a video object is obtained by segmenting it into AudioVisualSegments, which in turn can be described according to its creation and to temporal information.

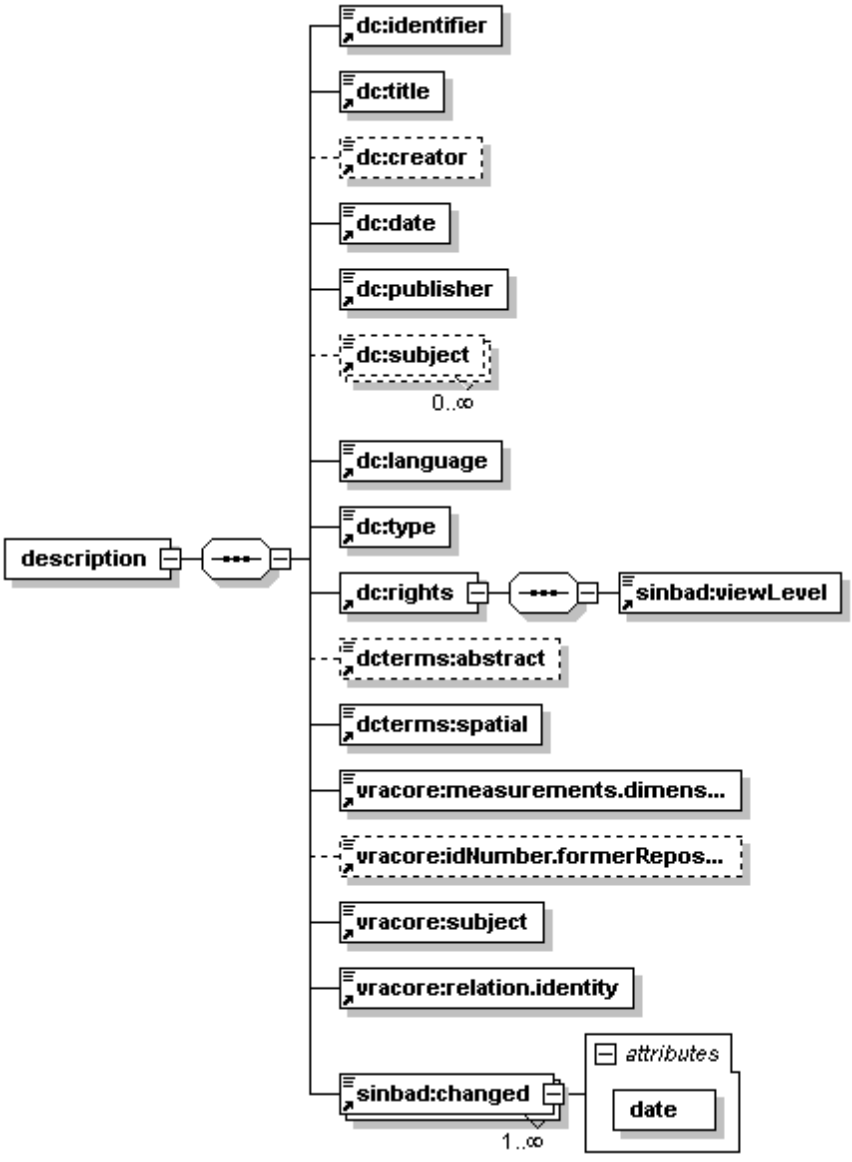


**Figure 3.5 – Metadata for graphical resources in SInBAD**

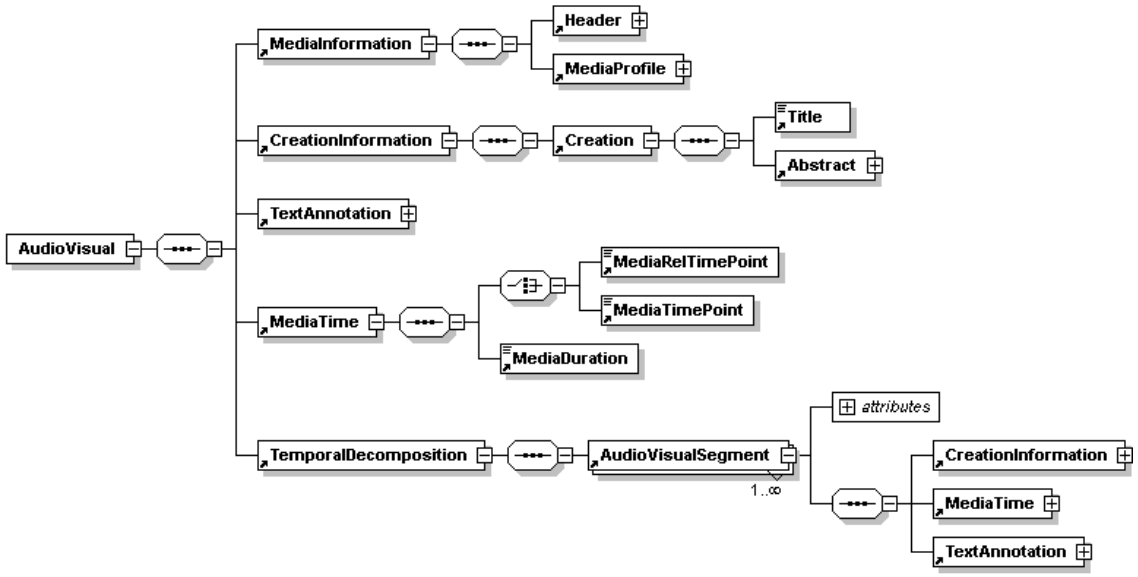


**Figure 3.6 – Subset of the MPEG-7 description standard**

### *3.3.5 Jazz*

The core catalog in the Jazz subsystem, Albums, was built around an existing database donated by the Portuguese jazz critic José Duarte. As result, its metadata is not represented in XML but in relational tables. Nevertheless, the most common fields can be mapped into Dublin Core and back when necessary. Such mapping is not made without some significant loss of information (the identifiers of musicians, tracks and instruments, required for referential integrity in the database, is not mapped), but the most important descriptors are preserved in the operation (e.g. album name to dc:title, album year to dc:date, and musician name to dc:creator).

The Jazz database is a rather complete information source – with about 20 tables relating entities such as albums, books, magazines, and labels – and is too

complex to show a diagram here. A simplified entity-relationship model[2] of a subset of the database diagram (the albums) is depicted in Figure 3.7.

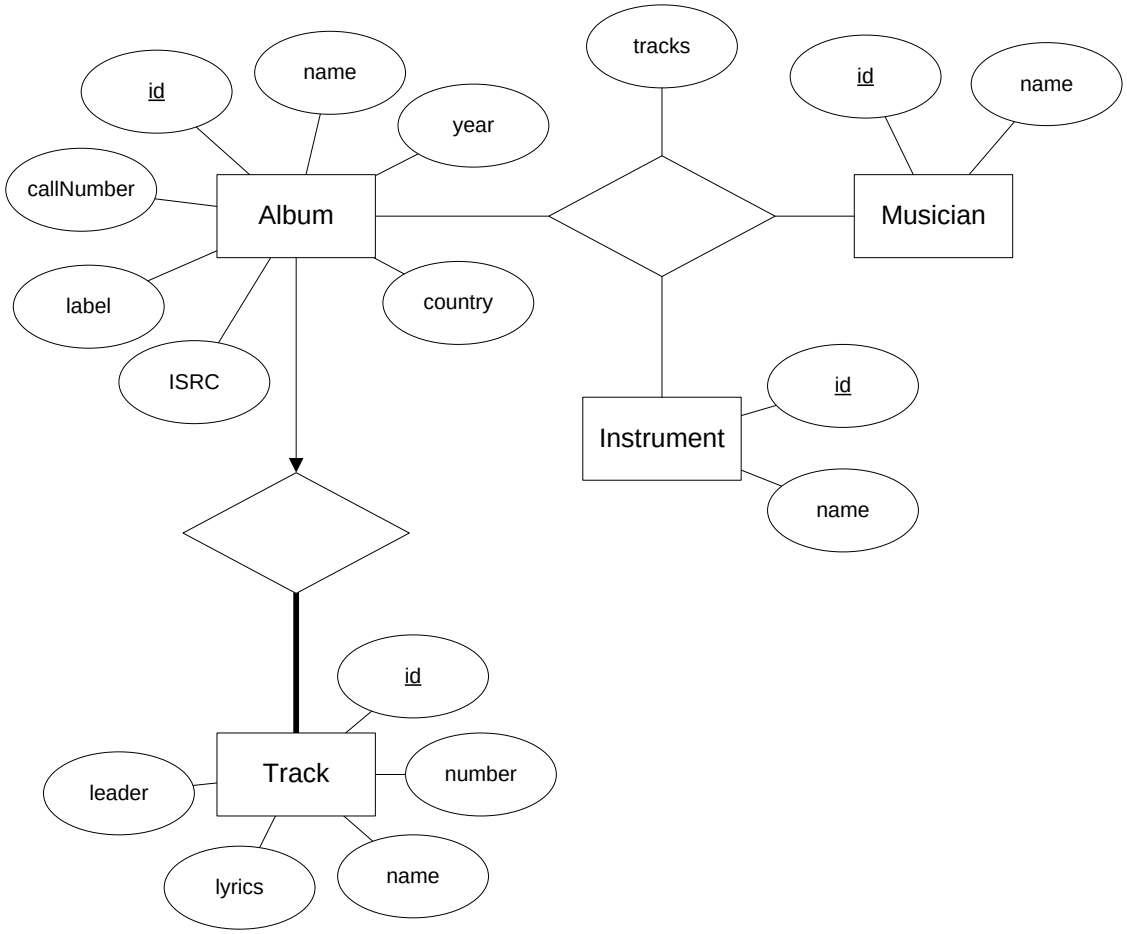


**Figure 3.7 – Jazz database simplified entity-relationship model**

### *3.3.6 Museum items*

The items in the collections are described using Dublin Core and a custom schema defined by the Portuguese Museums Institute. The metadata information is extremely comprehensive and includes descriptors for an item's authorship,

[2] In such diagrams, entities are represented by squared boxes, its attributes by ellipses and the relationships between entities by diamonds.

historical track, employed techniques, composing parts, past exhibitions in which it was features, among others. Figure 3.8 depicts a small subset of these descriptors.

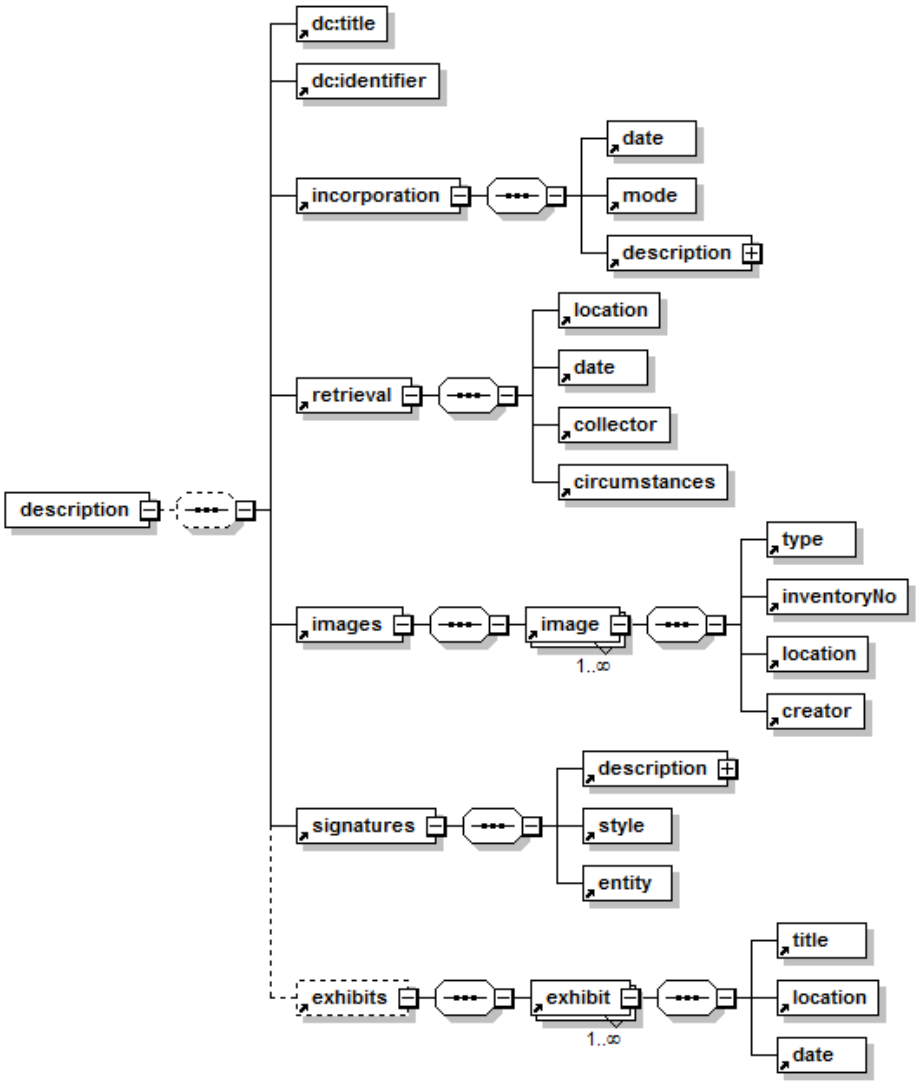


**Figure 3.8 – Subset of the metadata schema for museum items**

## 3.4 ARCHITECTURE

Figure 3.9 shows the architecture of the SInBAD system and some of the services it uses (only the Library and the Jazz subsystems are shown for clarity

purposes). On top of the stack, the SInBAD portal is the main website for the system, and it provides an OAI-PMH interface. This website may get data from each subsystem's Web Server (e.g. on searches), but users will mostly access objects through each one's website. Each subsystem, on the other hand, uses the DisQS module (details in the next chapter) to retrieve data and metadata from a distributed repository, while authorization rules are stored in a shared relational database. Some utility and external Web Services are also used to obtain some functionality.

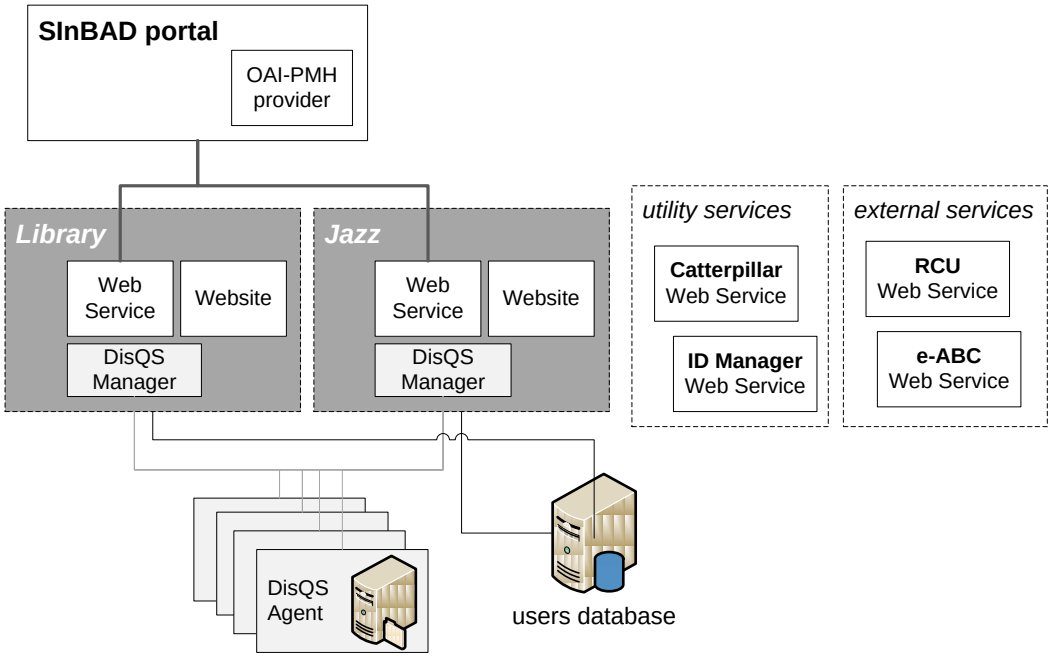


**Figure 3.9 – SInBAD architecture**

### *3.4.1 DisQS*

Before SInBAD, researchers from the Digital Libraries laboratory in the Instituto de Engenharia Electrónica e Telemática de Aveiro designed and developed some information systems such as the Portuguese Parliamentary Records Digital Library [116] and the Audiovisual Archive [117]. A common

characteristic in all systems is the large amount of data that needs to be stored, queried and retrieved.

Research inside the DL group led to the conception of DisQS – Distributed Query System [32][30][31]. The objective of DisQS is to allow a standardized interoperation between network nodes, which is accomplished with the interaction between the Web Services of a DisQS Manager and those of at least one DisQS Agent. The architecture of DisQS is depicted in Figure 3.10.

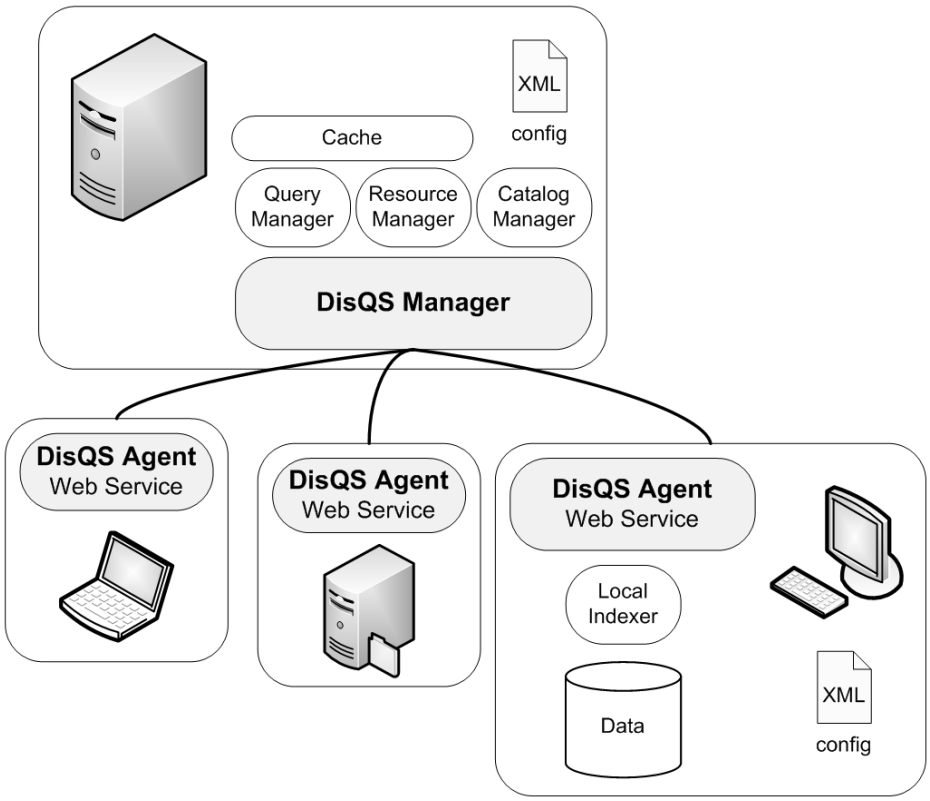


**Figure 3.10 – DisQS architecture**

The Manager has a cache which enables it to quickly respond to previous requests (search queries) and is further composed into three modules. Query Manager performs the distribution of search queries to the Agents and optionally filters results. Resource Manager is responsible for retrieving documents from and uploading to Agents; it can be configured to upload documents redundantly to

several Agents. Catalog Manager is responsible for inspecting a XML configuration file to determine which remote repositories must be accessed; it also controls updates in the network and manages the catalogs and its properties. The structure of the configuration is depicted in Figure 3.11 and Figure 3.12.

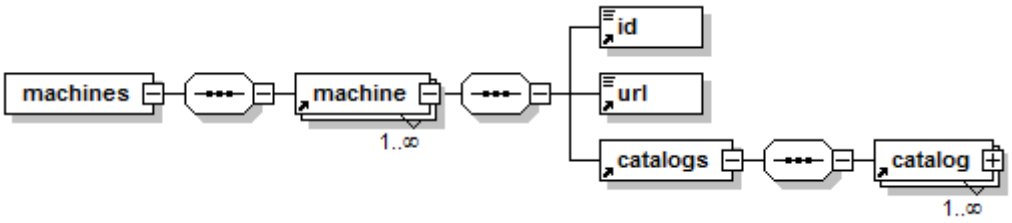


**Figure 3.11 – DisQS Catalog Manager configuration**

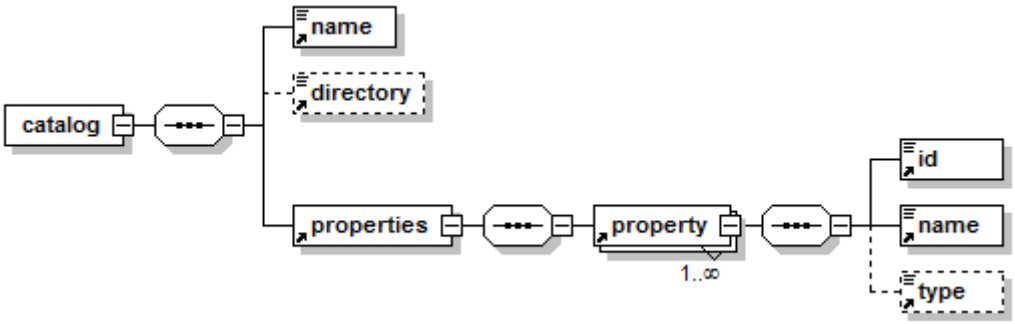


**Figure 3.12 – DisQS catalog configuration**

Each Agent is registered in the Manager, with an entry composed by its identifier, URL (of the Web Services), and a list of catalogs.

The catalog definition exists both inside the Manager's XML configuration file and at providing Agents. Each catalog has a unique name, a root directory (optional and only at the Agent) and a group of properties. These properties – described by a local identifier, name, and data type – define which terms in the metadata are searchable and/or retrievable in search results. A standard catalog

may, for instance, be pre-populated with the 15 terms from the Simple Dublin Core schema.

Inside each Agent, a Local Indexer module acts both as a wrapper to a search engine and as a manager for the catalogs and properties defined in its local XML configuration file. For instance, when a new document is created or uploaded in a catalog, it is up to this module to extract the metadata specified for the catalog and index it. It is also this module which interacts with the search engine in order to reply to search queries. Tests were conducted using Microsoft's Indexing Engine [118] and Swish-e [119], and SInBAD uses the former. Indexing Engine automatically indexes properties for some common file types (HTML, TXT) but to make it index our metadata files a two-step procedure was conducted:

- The QuiLogic's Ifilter [120] was installed in order to make the engine capable of indexing XML elements and attributes in a configurable way (by default the engine will only index the elements contents);
- To make the engine distinguish between these XML metadata files and other XML files that may exist, the system description files were given the extension "mtd"; the rules defined in the previous point could then be applied only to this extension.

Regarding the Web Services, both the Manager and Agents have an identical interface, which provides the most common file operations (GetFile, UploadFile) and methods to do a repository-wide search or only in specific catalogs. There were however introduced some methods to better distribute computing tasks. For instance, getting an image file from the repository and creating a thumbnail is a common operation in most digital libraries. We therefore introduced a GetImageFile method – which accepts the file identifier and the desired dimensions and format – thus making such image processing a distributed process, to be executed by the image owner Agent.

The SInBAD digital library was designed on top of the DisQS system, hence taking benefit of the following features:

- Distributed solution based on modular and interoperable design;
- Transparent access to a distributed file repository as if it was a single one;
- Configurable automatic replication of data;
- Load balancing, either by using a replicated data scenario where the provider (Agent) may be chosen dynamically, or by delegating computationally intensive tasks (image processing);
- Remote catalog creation and management.

### *3.4.2 A SInBAD subsystem*

The core of each subsystem is a Web Service which acts as a client to the DisQS system. In the case of the Library, for instance, Agents with at least one of its catalogs (Theses, Books, Publications, and Magazines) are used with search or file requests.

All Web Services have a common set of operations (e.g., Authenticate, GetCatalogs, and Search) and others specific to each catalog (e.g., GetThesis, InsertPhoto). These methods not only perform the communication with DisQS, but they also handle communication with the external services overviewed in section 3.2, namely the RCU which is common to every subsystem.

#### *3.4.2.1 Authentication*

While access to the WSDL of the Web Service of each subsystem is open and unrestricted, several methods can be only used by providing a security token. Such mechanism works in a way very similar to how browsers send cookies between a website and a user computer.

Before any of such secured operation is used, the **Authenticate** method must be invoked with the RCU login and the encrypted password. If successful, the Web Service initiates session variables and generates a token to be returned in the HTTP response. Subsequent requests made by a client must be accompanied by such token.

The authorization process, on the other hand, is completely handled by each application in the campus. SInBAD enforces a role-based authorization

mechanism on a per-catalog basis. Thus, users from the CEMED centre can be granted editing permissions on videos, while those from the External Relations Service can only create and modify photographic records.

#### *3.4.2.2 Website*

Every subsystem has a frontend where users can browse, search, and view the documents in the repository.

Both simple (quick) and customized advanced search forms are available for each catalog – some fields may only be queried in a subset of the catalogs and terminology may differ. A search is processed in the following way:

a) the subsystem's Web Service Search method is invoked with property-value pairs (ANDed);
b) the Web Service submits the query into the DisQS Manager;
c) the Manager determines which Agent(s) support the catalog(s) and dispatches the query;
d) each Agent uses its search engine to retrieve the results and responds back to the Manager;
e) which in turn aggregates results, that are returned up until the website.

In the case of simple searches, the Web Service uses pre-defined set of properties (the most common), assigns them the query value and in this case sends it to DisQS to be ORed.

While a different viewer is used for each type of document, a common set of graphical and layout rules are enforced in order to conform to the University's institutional image. The common and most prominent element in all websites is its header, provided by the Banners service, and consumed using the JavaScript client in a REST style. Each catalog has its own set of images configured in this external service.

Regarding the media viewers themselves, they take advantage of all the structural information obtained from the XML description. For example, users can read through a dissertation page by page or quickly navigate into a specific

chapter (as defined in the table of contents). Identically, users can either watch through an entire “3810” program or only one of its segments.

### *3.4.3 SInBAD portal*

Each subsystem has its own repository, Web Service, and website, and can be considered independent and autonomous in almost any aspect. Nevertheless, the SInBAD portal is the main entry point for visitors and editors of any catalog. Besides linking to the available catalogs, it serves two important purposes:

- **Search aggregation**. The SInBAD portal provides a global and generic search that spans the entire repository. It accomplishes that by querying the Web Services of all its subsystems and aggregating and uniformizing the results.
- **Authentication**. SInBAD implements a single sign-on mechanism – all users login at the portal and their credentials (HTTP cookies) are transparently used by all subsystems.

#### *3.4.3.1 OAI-PMH*

In 2001, one protocol emerged to promote the standardized interoperability between digital libraries and archives: OAI-PMH [13]. This HTTP based protocol describes the interface a repository must provide (the Identify, GetRecord, ListIdentifiers, ListMetadataFormats, ListRecords, and ListSets methods or “verbs”) and defines the XML structures metadata must be exposed in. Dublin Core is the recommend standards for describing resources and in SInBAD only those terms are exposed in this interface.

Figure 3.13 depicts the OAI-PMH model for a GetRecord response. Both the responseDate and the request structures exist in all responses and in this case contain: the date the response was generated, the verb (GetRecord), the metadata standard (oai_dc for Dublin Core), and the unique identifier of the document requested. The record retrieved contains a header (with the resource’s identifier, date, and catalog) and the metadata itself, exposed inside the oai_dc:dc

element (most Dublin Core terms from the repository XML are copied without changes).

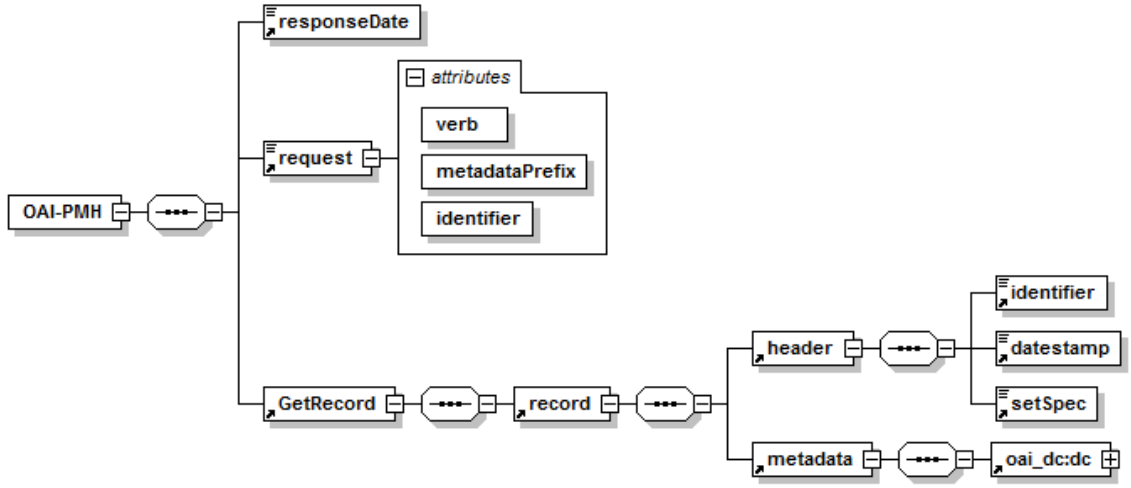


**Figure 3.13 – OAI-PMH GetRecord structure**

### *3.4.4 Utility services*

Several tasks involved in the creation and dissemination of digital objects in a digital library are time and CPU-intensive. Images must be converted and resized to different formats and dimensions, videos must be split, and text must extracted from documents, and all these operations can occur at external service providers rather than on the servers where the websites run.

A utility service – the Caterpillar Web Service – was therefore created to perform such operations. Once installed in a network connected computer, the subsystems can delegate such intensive tasks to any of these Web Services. Another advantage is that, since almost every such task occurs in submission processes (such as inserting a new photograph or book), time constraints are usually less strict than those involved in the dissemination of information by general users. Hence, commodity computers which are idle most of the time can be used.

A decision on the system design was also made early on which required that every object had a coherent but unique identifier throughout the entire distributed repository. The concept of an ID Manager was then created and later implemented as Web Service. This service can maintain numerical counters for unlimited scopes (i.e. applications), each with its defined format. SInBAD uses it with a size of 10 and the optional year attribute enabled (e.g. the ID 2008000373 was the 373th generated in the year 2008).

### *3.4.5 Interoperation with external applications*

When data and metadata must be aggregated from external applications, the question on which methodology to adopt arises [35]. A common strategy used by content harvesters is to periodically collect (harvest) the data from the known providers and save it (usually merged) in a local database. When these harvesters are later used for search purposes, they can provide results quicker since data is already prepared. The disadvantages of such methodology are however clear: data can quickly become out of sync, which can produce unexpected results such as the existence of multiple copies of the same object, or search results with non-working links. SInBAD retrieves information from other systems with a dynamic approach, thus querying its services in real-time.

#### *3.4.5.1 E-ABC*

As seen in section 3.2.1, e-ABC maintains an updated index of bibliographic references to scientific publications: articles, books, reports, and other material whose authors include teachers and researchers. No digital copy of those publications, however, was available to users, since this was not the purpose of the system.

With the development of SInBAD, this scenario has changed dramatically. Both e-ABC and SInBAD provide a Web Service by which the systems communicate. Whenever an author or an institution anchor (on behalf of the authors) inserts a new publication on e-ABC, he may also upload the corresponding PDF file. This file, however, is delivered at SInBAD's Web Service, along with the e-ABC generated identifier, which processes it (with Caterpillar) and

stores it in the Library's repository. If this operation is successful, e-ABC stores the URL for the document in its UNIMARC description file, in the format http://sinbad.ua.pt/publicacoes/*[identifier]*.

When users search in e-ABC, each result whose digital manifestation is stored in SInBAD provides a link to it (in the above format). On the other hand, when users search for scientific publications in SInBAD, the system actually performs the search in eABC's Web Service, limiting the results to digitized entries. When accessing the digital document, SInBAD fetches the UNIMARC description from e-ABC and displays it along side with the images (converted from the PDF).

Finally, SInBAD provides an OAI-PMH interface to e-ABC. It uses its Web Service to fetch all digitized records, converts them to Dublin Core, and formats the output to OAI-PMH compliant XML.

#### *3.4.5.2 Aleph*

Since the bibliographic references for most of the existent material in the University is stored in this library's system, it would make little sense to ignore it and require the time consuming task of re-cataloguing documents (or even copy-pasting the metadata). With this in mind, every developed back-office application has an Aleph communication module.

This module receives an identifier from the application (such as a system identifier or call number) and fetches the correspondent UNIMARC metadata description from Aleph. This description is presented to the administrator (or editor) in the application's interface so he can check the correctness of the information. When everything is validated and other tasks specific to the media are completed (such as specifying the table of contents and its permissions), data can be submitted. The back-office will transparently translate the UNIMARC into Dublin Core and whichever other description standard.

## 3.5 SUMMARY

The SInBAD system has become (in conjunction with e-ABC) not only the University's institutional repository, but also the entry point for users (internal and external) to access the digital content provided by several distinct sources.

By analyzing the features and capabilities offered by the system, namely in comparison with those obtained from popular digital library management systems, we believe the objectives were met:

- The flexible description model allows the system to fully support virtually any multimedia object, not only as storage tool but as customized viewing application; also, metadata can be exported both to Dublin Core and more specific standards;
- Its high granularity allows for both generic and repository-wide, or complex and customized search queries to be made, specific to a given media type;
- Such granularity is also used to control access to objects or some of its parts, thus better aligning with copyright issues;
- It is not an isolated application but instead one that provides and consumes information to and from other applications; its interoperation with e-ABC, for example, allows integrated views of scientific work and corresponding production reports, authority indexes, and the direct association with users, with the possibility to view the actual publications;
- It is Web Service based, therefore facilitating a standard based interoperation with other applications;
- Its modular design allows transparently replacing components or methodologies, such as the search engines, the storage functions, or the tools used in document processing;
- It is a distributed and scalable solution that can take advantage of remote resources.

Nevertheless, some issues and opportunities were identified at this point which could be improved in existing system by applying the most recent computational models.

The first one is concerned with the definition of workflows definition and execution. While most processes are service-based, they are currently hard-coded into the Web Services logic. Several of these processes could benefit from being described with a business process language such as BPEL and run with a business process engine.

On the other hand, since digital libraries store very large amounts of data, the decentralization could be taken to an even higher degree by collaboratively handling that data using peer-to-peer networks. This could prove to be especially important in an institution with hundreds to thousands computers idle most of the time.

In the next chapter some improvements to the digital library architecture are discussed and proposed.

# CHAPTER 4 – A SOA and P2P based architecture for digital libraries

## *4.1 INTRODUCTION*

In the last chapter the conception, designing, and implementation of the SInBAD digital library were discussed. While its architecture does comply with the initial objectives, a large computational capability remains nonetheless underused. The campus network is composed by thousands wired computers and hundreds connect to it wirelessly every day.

First of all, this represents a very large distributed storage space which can be tackled basically by any P2P product. This is not, therefore, an innovation per se. One has nevertheless to properly store resources and make them discoverable in light of the requirements of a digital library system. This leads to discussing the models, storage, indexing and searching of metadata of heterogeneous resources.

While the advantages of having a virtually unlimited storage space become easily apparent, such large number of mostly underused personal computers may

offer other resources such as computational power. In fact, the most time-consuming and computationally intensive tasks involved in the normal operation of SInBAD relate to the submission of new digital objects. In such tasks, it is the document processing services which become the bottleneck of operations, with CPU processing power being used to its full capacity.

The proposed architecture in this chapter therefore aims to tackle such distributed resources – both storage and processing power – using an underlying P2P framework.

There was identified one further refinement that could be made to such (business) processes. While they are currently Web Service based, its composition is hard-coded into the logic of each subsystem. There exists no declarative execution flow but rather a sequence of instructions with data preparation and service calls. An execution language such as BPEL can thus improve the modularity and flexibility of system, introduce a standard mechanism to compose and invoke services, and allow for changes to be made without the need for recompiling source code.

Furthermore, if we can properly integrate BPEL and peer-to-peer networks, a great number of advantages inherent from P2P will become available for the execution of business processes:

- Service availability can be largely increased by replicating services in several peers;
- Also, services located in computers behind firewalls and NAT systems can become reachable;
- Previously unused machines can host services to be used in an orchestration; idle times can also potentially be reduced;
- Dynamic service discovery and assignment in the P2P network can increase the flexibility and fault-tolerance of processes;
- Delegating part of the orchestration to other engines can help reduce the data on the messages transferred in the network and improve the overall performance.

The scope of this work is about analyzing how each of these characteristics and behaviors could be used to improve the efficiency of a service oriented environment. Specifically, a group of proposals are made on how business process management and execution software can become more efficient. These cover both architectural and small, localized, changes.

## *4.2 ARCHITECTURE*

The proposed digital library architecture is based on two fundamental concepts: P2P and SOA. Figure 4.1 depicts the architecture from a node point of view. While both storage and services are distributed, the P2P and Service modules make sure a node transparently accesses those resources.

**Applications** created for such digital library interact with the Service module. Examples of such applications include service registries, service composition and, more specifically, enhanced BPEL execution engines (**BPEL-e**) may also exist (as can a regular one) in some peers. Such engines are discussed further ahead.

The **Service module** abstracts the digital library implementation details and storage specificities by transparently providing interfaces to both services and data repositories. Every application's functionality relies on one or more Web Service available at this module.

The **P2P module** is built on top of the JXTA framework, which provides the basic P2P communication mechanisms. In JXTA, super-peers are named rendezvous peers and others are edge peers. If communication with peer groups from other networks is required, at least one relay peer must provide the outside communication.

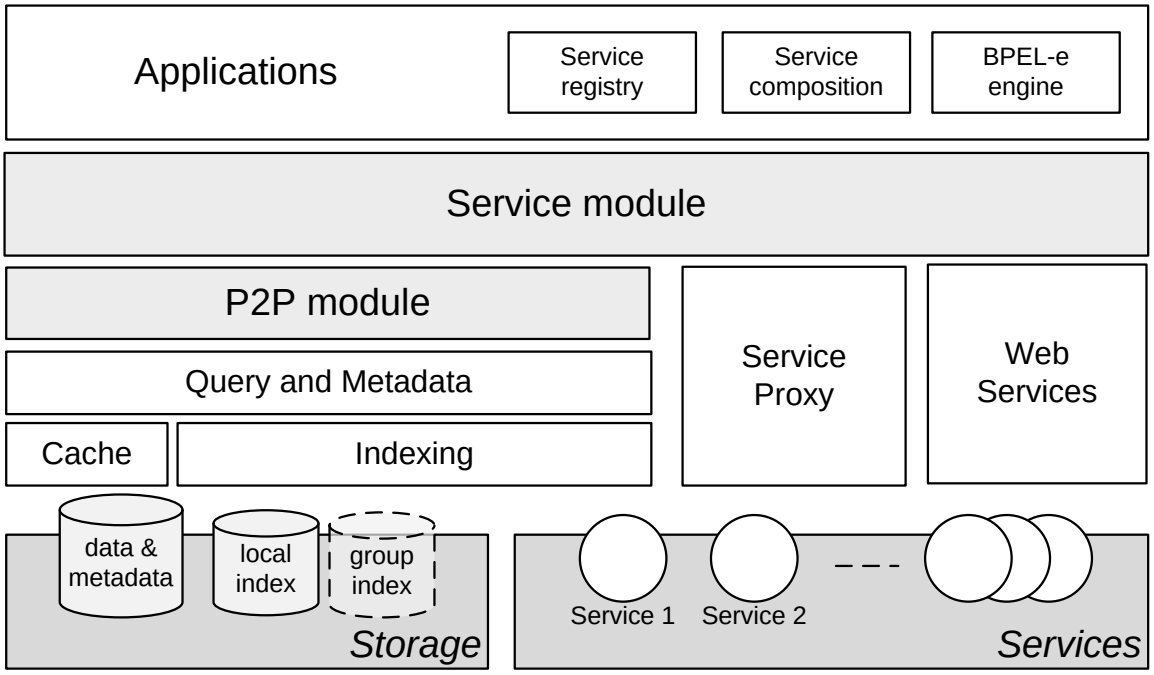


**Figure 4.1 – Digital library architecture based on SOA and P2P**

The **Query and Metadata** module prepares and pre-processes search queries before delivering them to the search engine. It is also responsible for processing results (ranking, description uniformization, etc.). Due to the modular design of this architecture, it is possible to change the search/indexing engine being used. Nevertheless, the query syntax used by this module must be coherent among all peers; otherwise proper communication between P2P modules would not be possible.

**Indexing** serves as a wrapper module for the search engine used in the digital library. As discussed earlier, it is desirable to make use of index/search engines in order to achieve high performance queries. This module creates a local index for the metadata of the resources it holds. In case of super-peers a group index is also created for the indexing of child nodes.

To improve the performance and responsiveness of applications a **Cache** module is also included. This may store data about previous search queries and information about other peers (neighbours, super-peers, etc.)

On the services side, the **Web Services** block implements a basic SOAP Web service client for regular HTTP services. The **Service Proxy**, on the other hand, interoperates with services available across the P2P network and otherwise unreachable. This proxy will be discussed further ahead.

### *4.2.1 Networking*

Figure 4.2 shows how network communication occurs between the peers. Two interfaces are available at the peer – P2P communication or service –, and both use XML as the message format.

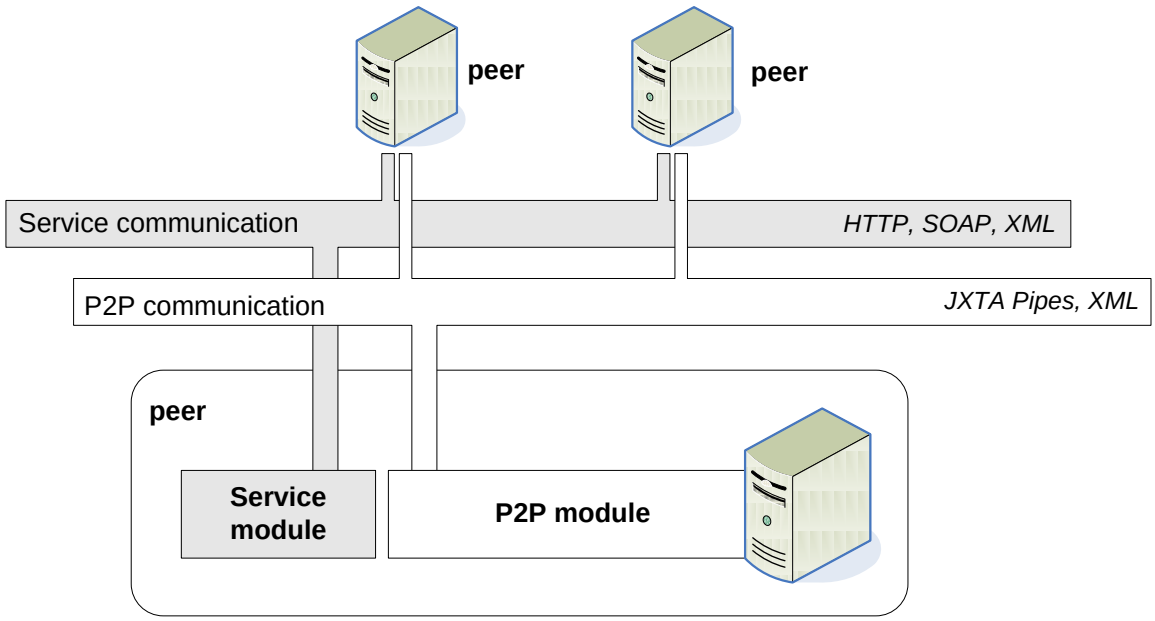


**Figure 4.2 – Networking of service-enabled peers**

The Service module performs all network communication related to Web Service description (WSDL) and invocation. Hence, this communication channel operates over the HTTP protocol using SOAP and XML.

The P2P module, on the other hand, handles P2P communication between nodes. With JXTA, this is accomplished with the generic concept of JXTA pipes, which are virtual connections between two endpoints [121]. A peer endpoint is a

logical abstraction of an address on a network transport capable of sending and receiving messages. Pipes enable the transparent connection of two endpoints which can be on distinct networks and use different transport protocols. Other than the messaging capabilities, no assumption is made about connectivity. Pipes also transparently handle the communication's routing process, which may require intermediary peers to accomplish, and deal with the details of message delivery between firewalled machines.

Data is sent over pipes using the concept of message, a set of named and typed contents called elements (i.e. name/value pairs). Two standard message encoding formats are used by JXTA messages: XML and binary.

XML is used for most of the communication between peers, especially for text-based messages. However, since some transports cannot transmit raw binary data, XML may also be used to encode binary content.

The JXTA Binary Message Format is designed to facilitate the efficient transmission of data between peers. Encoding large binary files in XML to be sent to other node can be both computationally intensive and require a much higher network bandwidth. If the underlying network transport supports binary data this is the preferred format for those scenarios.

## *4.3 P2P*

An array of autonomous nodes (the P2P network) is capable of performing the data storage required by digital libraries. For this purpose, and following the discussion from section 2.3.3, hybrid topologies offer greater flexibility, as they offer much of the centralized robustness while maintaining the advantages of the decentralized topology.

This architecture is designed to take advantage of such hybrid topologies. In each peer group (small institutions, departments, I&D units, etc.) a super-peer maintains an updated index of the group resources (documents and services). Other peers periodically send their local indexes to be merged at the super-peer (in the group index). When requesting a file or service provider, peers start by

inspecting their local cache and, if no match is found, a query is made to the corresponding super-peer. Each super-peer may also forward queries to other super-peers.

### *4.3.1 Metadata*

Most P2P applications enrich data with very limited metadata and restrict search queries to simple and common criteria, such as filename or file type. This leads to a higher simplicity in development and query processing, and reduces communications package size and memory required to store indexes.

Full-featured digital libraries, however, place much more complex requirements on its infrastructures. Using proper metadata to describe resources is crucial for digital libraries, not only when resources are presented to users but also when they are searched for. For example, it is commonly desirable to allow users to search for books by its author, musical tracks by duration, or movies by genre.

Metadata should also be coherent so that two peers can properly interoperate, since searching and requesting heterogeneous description languages is not a trivial task.

As discussed in the design process of the SInBAD system, we find it desirable to have two description levels: transparent to the format and format-specific. In the former, there are descriptors which can be used by almost any type of digital manifestation, such as title, author, and date. In the later, there are descriptors which are specific to the type of manifestation of the digital content, such as bibliographic references for books, or resolution for photographic data. The adopted metadata model therefore does not differ from those used in SInBAD: Dublin Core for the common property set, and a specific standard according to document type.

### 4.3.2 Indexing and searching

It is not efficient to search for file names or types in real-time, and querying for heterogeneous and complex metadata that way would not be feasible. Therefore some sort of search engine should be used.

A search engine is commonly based on inverted indexes, where words are stored (usually alphabetically, for better search performance) alongside with the identifiers of the documents where they were found.

To allow for phrase search or more complex queries – such as pattern-matching (wildcard expressions) or proximity expressions (one term near another) –, the positions of words in each document can also be stored. These are called inverted indexes because its structure is opposite to those of regular databases: words are the keys (what to look up), while documents identifiers are regular fields. When a query is submitted to the engine, the words are looked up in the inverted index and the documents matching all the words are retrieved.

In SInBAD the Microsoft's Indexing Service [118] was used in DisQS Agents for providing the indexing and searching capabilities. It became apparent, however, that it had several limitations and it was not adequate for this new architecture. Section 4.5 summarizes the analysis and evaluation of six search engines in the scope of a hybrid network. Results obtained from such evaluations indicate Lucene search engine to be currently the most suitable engine for this proposed architecture.

#### *4.3.2.1 Indexing configuration*

Since this new architecture is based on the same metadata models than those used by SInBAD, some configuration is needed to allow for each peer to know what terms to index and search for.

An identical mechanism was used with the DisQS distributed system, where XML configuration files with the indexable properties existed at both a manager and its agents. The adopted strategy in this architecture combines this approach with the results from previous work [122] based on JXTA handlers.

Upon installation, all peers can properly index and search a pre-defined set of file types and terms. This is done using message handlers, which associate a class (the handler) to a file extension. By default peers index the text from text (TXT, DOC, HTML, and PDF) and XML (only the content values) files into the index term “content”. Most importantly, they handle MTD files, which are XML documents reflecting the Dublin Core based metadata model. In this case, the search engine is instructed to index the basic Dublin Core’s 15 term set and the relevant terms (i.e. searchable) from the other standards.

On top of this basic configuration, new handlers can be later added to search-enable the system regarding other objects. For example, in a service-enabled peer, we may store the WSDL files of the provided Web Services in a special folder and create a handler to read the XML and index the namespace, the operations, etc. Should SInBAD be adapted to this architecture, and since metadata is supplied in separate files, only the handler for MTD files would be required for every peer. As a matter of commodity, however, handlers could be developed to automatically extract some specific features (e.g. duration in audio and video files or page count in PDFs).

Similar to the first version of SInBAD, where configuration files for indexing exist at both the DisQS Manager and Agents, super-peers should assure the propagation of the configuration files and required handlers to its nodes. This should occur both when a node first enters the network and periodically to keep changes synchronized. A schema identical to that shown in Figure 3.12 can still be applied.

#### *4.3.2.2 Querying language*

As it should be clear, the queries transmitted in messages throughout the network must be uniform in order for every node to interpret it. For a matter of simplicity a query language identical to that used by some search engines[3] can be

[3] See for example http://lucene.apache.org/java/2_0_0/queryparsersyntax.html

used. The syntax consists in a set of one or more <*term_name*>:<*value*> pairs, separated by Boolean operators. Grouping is achieved by using parentheses, phrase searches using double quotes, and term modifiers (fuzzy, proximity, and range searches) using wildcards. The following is a query example for our metadata model:

```
dc_title:Aveiro AND vra_subject:”Cartazes de desporto”
```

### *4.3.3 Topology*

In order to fully evaluate the benefits of a hybrid topology one must be aware that for this semi-centralized search to work, peer indexes must be stored at the super-peer. Such operation, roughly equivalent to a file transfer and a file merger, can become a bottleneck to the network if indexes are too large or if index updates are requested too often. An index with some gigabytes could take several minutes to be transferred to the super-peer.

Ideally, a P2P application should adapt to changing scenarios and to the dynamics of peers entering and leaving the network. Hence, the P2P layer should be configured to use a super-peer based topology but to fall back to a decentralized environment when required:

- When the network is first initiated and a super-peer is designated, the system should temporarily function in the decentralized mode while indexes are transferred.
- If a leaf peer has a very large index which was still not transferred, the super-peer may flag it so that it forwards search queries and combines results with those from its indexed nodes. These large file transfers can be scheduled to occur at lower activity periods.

A JXTA-based infrastructure was designed adopting these principles. In its normal functioning mode, networks are hybrid, with some of the nodes acting as indexing super-peers. However, in some occasions the network may fall back to a

(semi-)decentralized mode: in the startup process, when a node with a very large index enters the network, or if existing super-peers leave the network.

### *4.3.4 Optimization*

While the hybrid configuration offers more stability and robustness to applications, the weaknesses of centralized models can still be observed locally. A cluster of leaf nodes can be suddenly left orphan if the super-peer is shut down or leaves the network. While the network will fall back to a working decentralized topology, applications could benefit from a more efficient approach.

Nodes should thus be classified as peers, super peers or backup (B-) peers. Regular peers maintain a connection to the super peer and a reduced number of other peers. A super peer maintains connections to all peers in the network and an index of its resources (data and services). To ensure a high availability, we introduce a backup peer, which periodically fetches the connections and the index from the super peer. In case of failure of the super peer, the B-peer becomes the network super peer and instructs all peers to act accordingly.

Super peers and B-peers should be selected based on their hardware capabilities and an estimate of the uptime rather than on the amount of resources they offer. In fact, a super peer may not even share any data, since it is of greater importance that it replies very quickly to requests and forces the minimum network changes.

Within the organization, resource access can be optimized by periodically and automatically balancing the load: by moving resources or replicating them over nodes to maximize throughput.

If an organization has few computational resources, it can form a virtual organization with other trusted entity. On the other hand, if one organization has too many nodes to aggregate under a single super peer, it can follow a multi level scheme: one super peer will aggregate several other super peers, which are responsible for a subnet of the organization.

## 4.4 SERVICE ORIENTED ARCHITECTURE

### 4.4.1 Dynamic Service discovery

In a widely distributed and heterogeneous service oriented environment, several applications may make use of Web Services created by multiple developers and hosted at several providers. However, in order to take full advantage of the potential of such services, the proper service must be found at the right time. Having some sort of discovery mechanism in such environments is therefore a desirable feature.

The importance of service discovery is better thought in the two different application stages. In a first stage, at design time, developers must have the necessary tools to find existing Web Services (either internally within the organization, or externally in the Web or at some business partner) in order to reuse existing components, thus boosting productivity. When their applications are running in production, a different requirement can be placed on the discovery system: providers may become unavailable, new versions can be deployed, and a provider may be chosen from a list of complying parties.

Let one consider the case of a BPEL business process. The BPEL language is built upon Web Services and therefore uses the Web Service Definition Language (WSDL) extensively. In fact, both the partners (the service providers) and the process itself are exposed as WSDL services.

A simplified skeleton of a BPEL process definition is presented below[4].

```
<wsdl:definitions>?
  <!-- types, messages, portTypes, and parternLinkTypes -->
</wsdl:definitions>

<process>
```

[4] The pattern should be interpreted as follows: elements with an asterisk can have zero or more occurrences, with a plus sign have at least one occurrence, and those with a question mark are optional but non repeatable.

```
  <import namespace="URI" location="URI" importType="URI" />*

  <partnerLinks>?
    <partnerLink name="NAME" partnerLinkType="QNAME" myRole="NAME"?
       partnerRole="NCName"? />
  </partnerLinks>

  <variables>?
    <!-- the variables -->
  </variables>

  <sequence>
    <!-- the activities -->
  </sequence>

</process>
```

The process definition starts by declaring the WSDL types, messages, portTypes and parterLinkTypes involved in the activity execution. Namespaces are then imported, the partner links and its roles defined, and the variables declared. Partner links are instances of typed connectors specifying the WSDL port types involved (see section 2.5.1.1). Roles define the services the process will use, and a myRole is a service provided by the BPEL process itself. Variables are used to contain data (WSDL messages, or XML schema types or elements) in a process and they constitute its state during runtime. Only then the actual process activities are defined within the <sequence /> element.

When a service provider hosts a Web Service, it makes its WSDL definition available at some URL. By inspecting this WSDL, we can find something similar to the following XML elements:

```
<wsdl:service name="myService">
   <wsdl:port name="myServicePort" binding="tns:myServiceBinding">
      <soap:address location="http://example.com/Serv" />
   </wsdl:port>
</wsdl:service>
```

Usually, before execution, a BPEL engine will compile all definitions (BPEL and WSDL) into a runtime, and the service addresses become coded. A BPEL engine could however use a discovery service to find providers hosting the same service – with identical WSDL definitions but obviously with different

<soap:address /> elements. Such a simple modification in the behavior of a BPEL engine makes the process execution more flexible, as it is no longer tightly bound to specific providers. As a consequence, services can be dynamically chosen to improve flexibility, fault tolerance and throughput.

A possible approach is to use service registries to provide such discovery mechanisms.

#### *4.4.1.1 UDDI*

There are two main service registry standards – ebXML and UDDI. ebXML [123] or Electronic Business using eXtensible Markup Language was created in 1999 as a joint initiative between OASIS and the United Nations Center for Trade Facilitation and Electronic Business. In 2000, a consortium led by IBM, Microsoft, and Ariba created a platform-independent and XML-based registry for Web Services: the UDDI – Universal Description, Discovery and Integration. From the major software vendors only Sun appears to support ebXML, while UDDI is supported by products from HP, IBM, Microsoft, Oracle, and Sun. Hence, UDDI remains the only standard which is widely adopted and can assure a high degree of interoperability.

The UDDI specification [124] defines a standard method for publishing (at UBRs – UDDI business registries), discovering, and managing information about Web Services, their providers, and technical interfaces which can be used. UDDI is itself a set of Web Services and is therefore based on standards such as HTTP, XML, SOAP and WSDL.

There are four core data structures in the standard, which express the relationships depicted in Figure 4.3.

The **businessEntity** structure describes businesses and service providers (enterprises, departments, or groups). This entity may include several names (one per language), contacts, URLs, descriptions, and classifiers according to some categorization system (in the **categoryBag** element).

A provider may contain multiple logical services, described by the **businessService** structure. Services are described with names, descriptions and categoryBag elements. The physical implementations of a service are not defined at the businessService element but rather using the binding structure.

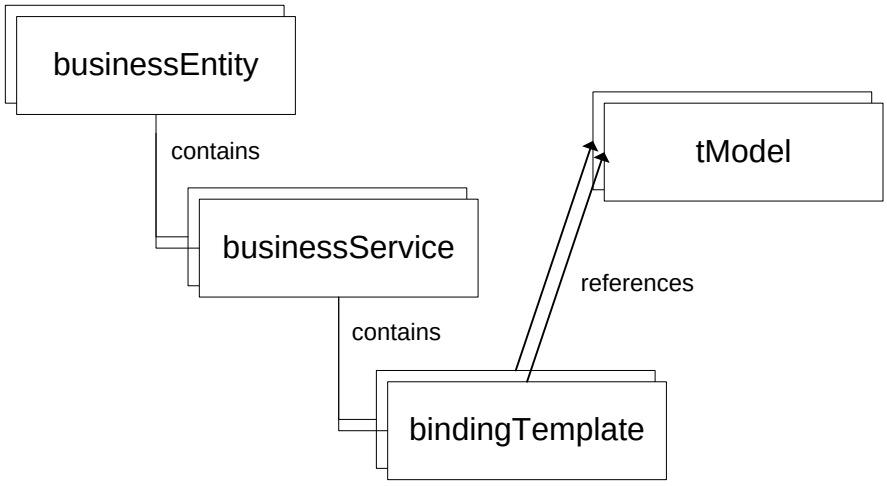


**Figure 4.3 – UDDI core data structures**

The technical information about a service is found in the **bindingTemplate** structure. Each of such elements describes an instance of a Web Service located at a specific network address, and may include descriptions and categoryBag elements. A bindingTemplate also contains details for each tModel referenced, along with description and documentation URLs.

Additional information about an instance of a service can be described using the **tModel**, an abstract structure which can represent any concept or construct. tModels are commonly used to link to WSDL documents, but may also be used to define transport protocols, security models, or categorize the service (using thesauruses, free-text keywords, etc.).

Following the discussion of the business benefits inherent from using a service discovery mechanism, the advantages of using UDDI seem apparent. First of all, it helps improve development efficiency, by providing the means to find

existing services, thus reducing the chances of duplicate development and reducing the time required to create new applications or services. Also, since technical documentation can also be found using a UBR, developers may more easily gain knowledge about the interface protocols and how to interact with the services. On the other hand, there is an increased manageability of the services created across the enterprise. A single view of different services, versions, legacy applications and interface formats is discoverable.

A document from Microsoft [125] published in 2003 describes the vision of this enterprise on the benefits of UDDI and uses three different business scenarios to illustrate them. The first one focuses on the developers and IT efficiency at design time; at this particular stage the Microsoft's current development IDE – Visual Studio .NET – natively supports UDDI services. The second shows the advantages of run time discovery, making applications more flexible and robust. The third scenario outlines how UDDI services can be extended beyond an enterprise to external business partners, allowing them to not only discover its services but also the knowledge that makes it easier to integrate.

The latest versions of the UDDI standard (3.0) have recognized the need for federated control in real-world scenarios and have therefore tried to offer more implementation options in order to better integrate into different network topologies. For that matter, registries can be of three types – private (corporate), affiliated, or public – that comply with the same specifications.

From the end-user's perspective, public domain registries act as an open and public service in a cloud. In this scenario, UBR nodes can automatically share and replicate data changes between each other. Administrative functions can nevertheless be secured.

A registry can also operate in a firewalled, private mode, allowing a corporation to manage and discover its own services isolated from the public network. There is no sharing of data with other registries.

Registries can finally be affiliated, a policy based configuration which allows for a controlled environment with access limited to authorized users and sharing of data in a controlled manner. This is one of the most important concepts introduced with the 3.0 version of the standard.

#### *4.4.1.2 Issues and opportunities*

UDDI seems the obvious technology to implement a service discovery mechanism in a network. However, a closer analysis of how to design the service discovery process in a P2P network suggests it is impractical in such environments.

Let one consider the design of a P2P based digital library. The first architectural issue to arise is where to install the UBRs. Due to the hybrid configuration of the network, the first natural choice would be the super-peers. However, what if a given super-peer leaves the network? If its leaf nodes cannot directly query – via Web Services – any other super-peer (which can actually be a common situation, since super-peers frequently act as relay nodes) they will become incapable of discovering services.

While federation and affiliation concepts introduced in the new version of the standard have given a broader range of design options, the physical implementation is still nevertheless basically static. Installing a distributed UDDI registry composed of dynamic nodes is a non-trivial task which could lead to unacceptable results. The highly dynamic nature of P2P could lead to a very large and frequent number of messages being transmitted to update these registries, a number which could exponentially increase if a replication mechanism was in place. Implementing a completely decentralized UDDI solution (with no super-peer) is also clearly impractical.

On the other hand, the dynamics of the network could lead to UDDI registries being filled with obsolete entries. When peers enter the network, they can automatically register the services they provide in a UDDI publishing node. However, if they later leave or are shut down in an uncontrolled manner (power loss, for instance), no entity would unregister the services the node provided.

Some sort of P2P notification mechanism would be required to prevent such scenario.

There are also several issues regarding UDDI registries which are common to any network topology. While XML and Web Service standards (SOAP, WSDL) have been widely accepted and adopted in the industry, UDDI has seen a slower and more limited acceptance. There were also indentified the following limitations [126]:

- The mapping of Web Service artifacts into UDDI is inappropriate. Since UDDI is not designed to store WSDL definitions, the current technical approach consists in mapping wsdl:portType elements into UDDI tModel entities. Consequently, the results of a search query for a specific tModel name do not include important information (such as the namespace, mapped in a categoryBag) and more UDDI requests are needed, reducing performance.
- No interoperability exists between service registries from different vendors, making it difficult to later copy data to a different implementation.
- There is no standard way of limiting the access to records in the registry; existing solutions are non-official extensions.
- The querying capability of the registries is very limited. There is the lack of the logical NOT operator, support for arbitrary combination of logical AND and OR operators, nested find_tModel queries, or group-by operators.

Despite the limitations of the UDDI registries and its querying capabilities, the standard provides nonetheless agreed upon schemas for registering services and their providers. Whatever discovery mechanism is created, these schemas can eventually be adopted (or exported into). More importantly, one should understand the advantages of service categorization (financial, mathematical, document processing, etc.) and categorization contexts for search decisions (service level agreements, localization information, etc.).

#### *4.4.1.3 Service discovery in P2P*

Traditionally, P2P applications were designed for file sharing purposes. Networks such as Gnutella, BitTorrent, and Napster are file oriented rather than resource oriented (files, services, etc.).

In the most common P2P systems, file properties (such as the name, size, and type) are indexed in order to speed up queries; in hybrid topologies, indexes from a cluster's peers are also merged into the super-peers indexes.

To make use of a P2P network for service discovery, we need an infrastructure which allows:

- Publishing and indexing service definitions;
- Querying for peers which provide specific services.

To accommodate these requirements, we can use JXTA, an open-source project which consists in a group of open and generic protocols to connect heterogeneous devices in a P2P network. This Java based framework aims the creation of an interoperable and platform independent P2P network. While JXTA protocols are not standards, they are XML based and therefore programming language and platform agnostic.

JXTA peers are known between each other through advertisements: nodes publish information about themselves (and eventually the resources they have) using Peer, Peer Group, Module Class, Module Specification, and Module Implementation advertisements.

WSDL definitions from service providers in a P2P network can also be published using advertisements; in the JXTA-SOAP project, for instance, they are encapsulated in Module Specification advertisements. For service discovery to properly function under JXTA-SOAP, such advertisements must include the service WSDL and a tag indicating whether a "secure" pipe shall be created. Apart from those two tags, one is able to provide additional information in a ServiceDescription class: properties such as a service's name, creator, version,

and description. As will be discussed in a following section, however, we chose not to use the JXTA-SOAP project.

#### *4.4.1.4 Publishing*

Publishing a service in peer's repository is a two step operation: 1) the description and technical details required to consume the service must be stored, and 2) metadata must be indexed to allow for an effective search. For both operations the requirements vary according to the type of discovery that will be used:

- **Design-time discovery**. At this stage, developers and system architects may need to find and gain knowledge on how to use services for a multitude of purposes: reuse existing functionality to speed up the development process, gain access to systems and resources managed by other entities, or communicate with services from business partners.
- **Run-time discovery**. Once systems are configured and in production, run-time service discovery can offer an increased degree of robustness and failure resilience, allowing faulty or inactive providers to be replaced.

Hence, for design-time discovery the (distributed) service registry must at least contain the WSDL definition documents and the access point network addresses. Additional information is however recommended to maintain a proper registry: details about the provider and its contacts, descriptions, URLs for more technical insight, and categorization.

Finding adequate service taxonomy (a hierarchical classification structure) is fundamental to enable developers to quickly discover suitable services to solve a specific problem (financial, mathematical, document processing, etc.) and is itself a complex research subject. It is difficult to adequately define a service because too many types of services exist. Although there are some popular categorization systems available for products and services, such as UNSPCS [127] – United Nations Standard Products and Services Code – and eClass [128], most are closed and too complex and product oriented to apply in this scenario.

Instead, the adopted taxonomy follows the methodology suggested by Richards [129], in which a hierarchy should be based on (or even using only) four basic SOA service types:

- **Business services.** These are abstract services derived from specific use cases or scenarios. In the case of SInBAD, for instance, this classification would be given to specific subsystem services (GetThesis, CreateVideo, UpdatePoster, etc.).
- **Enterprise services**. These are concrete services needed to implement Business services, usually in one (business) to many (enterprise) relationships. Despite what the name would suggest, they may or not be shared across the enterprise. From the SInBAD digital library we could include in this category the user-related services (CreateUser, AuthenticateUser, …) or fine-grained services that implement specific parts of a business service (such as InsertVideoFile).
- **Application services**. These are supporting services tightly bound to a specific application context, and therefore are not usually shared across an enterprise. They are generally used to perform fine-grained functions such as data validation, collection, and transfer. Examples of such services in SInBAD include CheckMagazinePdfExists, MoveVideoFile, and GetPublicationPageCount.
- **Infrastructure services**. These are business-agnostic services, typically shared across the enterprise and used by different applications in various scopes. Caterpillar, ID Manager, and even DisQS Web Services are examples of infrastructure services, since they provide functionality to any type of application.

Since this work is focused on harnessing the computational power and distributed storage space available in the P2P based service-enabled network (rather than trying to improve specific application services) we will only further categorize infrastructure services. And hence new categories were defined:

- **Storage and Indexing**. This includes methods for storing and indexing data and metadata associated with a particular application (although not in a way or place designed for that specific application); the idea is to have a common infrastructure available to any application;
- **Security**. This refers to services which provide security related (but application agnostic) functionality, such as encryption and hashing;
- **Document processing**. This relates to generic services whose function is to transform or extract features from documents; this category is further refined into four divisions:
    - Textual – refers to functionality related with text formats (DOC, PDF, RTF, TXT) such as format conversion, splitting, merging, but also the extraction of information (description from embedded metadata, page count, word count, etc.);
    - Imaging – all the processes related to transforming image files – format conversion, resizing, cropping, rotation and flipping, watermark embedding, and application of filters or styling effects – and extracting metadata from them;
    - Multimedia – identical to the previous category, this relates to the processing of video and audio files and obtaining embedded information.

The diagram of the adopted taxonomy is depicted in Figure 4.4. Two notes about the hierarchy: 1) while it was purposely kept simple, new blocks can be added if it necessary later on; and 2) if no subcategory is suitable for a given infrastructure service at a certain level, the “parent” classification should be used (e.g. a service such as ID Manager can be considered a Infrastructure service while not being adequately defined by neither of its subclassifications).

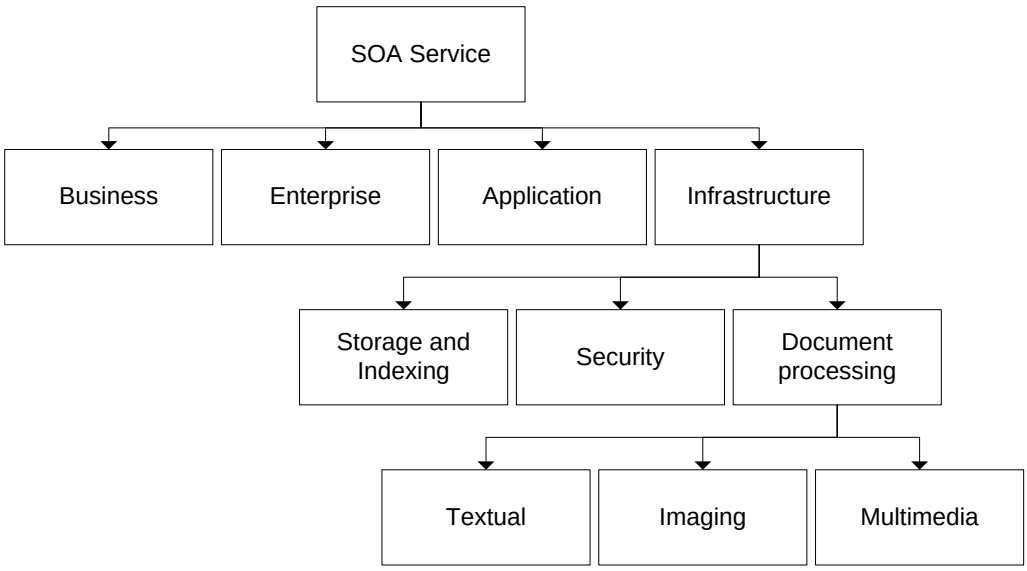


**Figure 4.4 – Service taxonomy**

Regarding runtime discovery we are first of all concerned in finding equal services from different providers, i.e. methods with identical signatures. There are a few valid options for choosing which values should be used in the service description (publishing) and in the queries sent to the network (discovery). Probably the most error-resilient method would be to hash the WSDL (without the <soap:address /> element, which may vary in different nodes) and query for services using those hashes. This could however be inconvenient both at the provider side (as more parsing and computing operations would be needed) and the consumer/application end (hashes would have to be either hardcoded or stored in a configuration file). It also assumes two providers must offer the exact same group of operations in its definition. Two providers with the exact same ConvertImage method would not be interchangeable at runtime if only one of them had a CropImage method, for instance.

A simpler option consists in using the targetNamespace attribute of the <wsdl:definitions/> element in the service WSDL. We would then use the declared namespace for any disambiguation, which is its purpose. Also, the service provider can easily publish this property, since it likely already has access to it or can

simply scan the XML to find it. On the consumer end, discovering services using an URI rather than a hash string is simpler and easier to remember.

There is however additional information that can be relevant to the runtime selection process: quality of service, pricing, localization, etc. Such added layer of intelligence is beyond the scope of this work.

#### *4.4.1.5 Indexing and searching*

We have identified the required and the recommended metadata fields to be indexed and discoverable in the network, summarized in Table 4-1.

**Table 4-1 – Service description elements**

| Field | Definition |
|---|---|
| **Identifier** (mandatory) | An unambiguous identifier of a service. This is the URI composed by the namespace and the operation name, i.e. the operation's **soapAction** attribute in the WSDL. |
| **Title** (optional) | The title of the service. |
| **Creator** (optional) | The author or entity responsible for developing and/or maintaining the service. |
| **Date** (optional) | The date or period of time associated with the resource. |
| **Description** (recommended) | A description of the service. This can include general information, service usage, or any other information deemed relevant. |
| **Subject** (recommended) | The service category, expressed using the proposed taxonomy. For multi-level categories, one can repeat this element as necessary (e.g. one entry for Infrastructure and other for Infrastructure/Security). |
| **Coverage** (mandatory) | The actual URI location of the Web Service. This can be a public URL or a private one (localhost), and one can retrieve such information from the **location** attribute of the **soap:address** element in the WSDL description. |
| **Type** (mandatory) | The fixed "service" value. |
| **Language** (optional) | If applicable, the language in which the service interoperates (data, interfaces). |
| **Publisher** (optional) | The entity responsible for the publishing or availability of the service. |

| | |
|---|---|
| **Relation** (mandatory) | The URI location of the Web Service's WSDL. |
| **Source** (optional) | A related service from which the described one is derived. |
| **Rights** (optional) | Information about rights associated with the service (property, intellectual, etc.) |

As it should be evident, the descriptors set is a subset from the Simple Dublin Core element set. The purpose of this methodology is to take advantage of the existing infrastructure: one can use a single index and the same indexing and search modules with the metadata of any kind of resource – documents and services – by creating, as observed in section 4.3.2.1, an index handler for WSDL files.

As noted in the table, only four fields are mandatory, and these are the ones we always have access to: any handler can extract the soapAction and soap:address from the WSDL file, the WSDL location is obtained when indexed (either locally saved or from an external provider), and the type is fixed. The other fields, however, are not commonly made available in the WSDL. It should be noted that while several initiatives [105][130] have appeared regarding the annotation of Web Services, they focus on the semantics of operations and data. That is an important research area, and one that can add additional layers of knowledge to this network; nevertheless, we have focused on providing additional description and authorship information for developers. We therefore propose using the ubiquitous <wsdl:documentation/> element to inject such information using Dublin Core XML. We can use it either at a document level inside the <wsdl:definitions/> root node or on a per operation basis inside each <wsdl:operation/> (the documentation element can also be used in other locations which are not relevant for this purpose). The two usage locations are exemplified below.

```
<wsdl:definitions targetNamespace=”srv://security.infrastructure.dl”>
      <wsdl:documentation>
            <dc:description>
                  Group of methods related to encryption and hashing
            </dc:description>
            <dc:subject>Infrastructure</dc:subject>
```

```
            <dc:subject>Infrastructure/Security</dc:subject>
            <dc:type>Service</dc:type>
      </wsdl:documentation>
      ...

      <wsdl:portType name=”SecuritySoap”>
            <wsdl:operation name=”Hash”>
                  <wsdl:documentation>
                        <dc:identifier>
                              srv://security.infrastructure.dl/Hash
                        </dc:identifier>
                        <dc:creator>Marco Fernandes</dc:creator>
                        <dc:date>Fri, 07 Sep 2009</dc:date>
                        <dc:description>
                              This method encrypts a text file using
                              the MD5 or the SHA1 algorithm.
                        </dc:description>
                  </wsdl:documentation>
                  ...
            </wsdl:operation/>
            ...
      </wsdl:portType>
</wsdl>
```

When indexing this file, instead of creating a single entry in the index, the handler will enter each operation as an individual “resource”. One should note that, when applicable, top-level descriptors should be inherited by each operation – the subject and type tags in the example above.

Providing such additional information requires, of course, that some IDEs and other development tools help developers in this task. Some already facilitate that job although by only accepting a generic text – Visual Studio, for example, allows a developer to document a method using an attribute with a **Description** parameter, which then appears in a <wsdl:documentation/> element:

```
[WebMethod(Description="This is the Encrypt method description")]
public string Encrypt(string input) {
      ...
}
```

### *4.4.2 Service invocation in P2P*

The typical development of a Web Service based application proceeds as follows: 1) a developer will add a reference (either manually or using a discovery service) to the services, which 2) creates proxy classes to interact with them and

3) stores the address URLs in a configuration file. Once the application is in its production stage, one can safely change the address to that on another machine provided its methods have identical interfaces.

This is where a dynamic discovery mechanism can be placed – if we can change the provider in runtime, we will augment the application's flexibility and robustness. The challenge however has now moved to the invocation stage: with the service-enabled P2P network we have created the possibility of having a wider range of providers, but if a peer can only connect to another through the P2P network, how will it interoperate with the second's Web Service?

#### *4.4.2.1 JXTA-SOAP*

The JXTA-SOAP project aims to solve that exact question. JXTA-SOAP is an add-on to the JXTA framework which allows Web Services to be invoked in the P2P network by transmitting SOAP messages using JXTA pipes.

A Web Service is first made available in a peer by creating and publishing it with Axis [131], an Apache SOAP engine written in Java. Advertisements are then created and sent to the network, containing the WSDL of the service. When other peer needs to invoke that service, a Java proxy client is then created.

While this seems to solve the P2P invocation problem, there are several disadvantages in the approach used by the JXTA-SOAP API:

- Since Web Services must be created and published with Axis, one is obliged to only use Java based services (unlike the JXTA framework, whose API is available in a variety of programming languages);
- The transparency is lost with the creation of this additional layer: instead of calling a Web Service, one will have to now call a Java function;
- This also is very limitative, since it makes all existing non-Axis Web Services useless unless an Axis proxy is made for each of them with an identical interface;
- Finally, replacing the Web Service logic with this add-on neglects the fact that two peers can sometimes interoperate outside the P2P network, i.e.

using regular HTTP request. Peers are not in every occasion behind a firewall or NAT systems, and in this particular scenario they may frequently be in the same LAN. Going through the P2P network when a simple HTTP request could be made would only add further serialization to the process.

#### *4.4.2.2 A simpler approach*

A closer look to what happens under the hood hints on another direction. The JXTA-SOAP modules are responsible for creating the proxy classes and sending the service advertisements to the network. The actual service invocation, however, is a very simply process which splits SOAP messages into packets (the JXTA network has a message size limit) and sends from the client to the server peer using pipes. On the server side a module regroups the message, invokes the Web Service, and replies in a similar fashion.

One wishes to both enable Web Service invocation over the P2P network and to overcome the issues summarized in the previous section. The solution is depicted in Figure 4.5. Peer 1 is running an application which is client of services with the same interface of the Security Web Service, available at Peer 2. If this service is publically available and the two peers are within reach, direct HTTP requests can be used to invoke it (solid line, black). If, on the other hand, they can only communicate using the P2P infrastructure, a Web Service proxy is used (dotted line, gray).

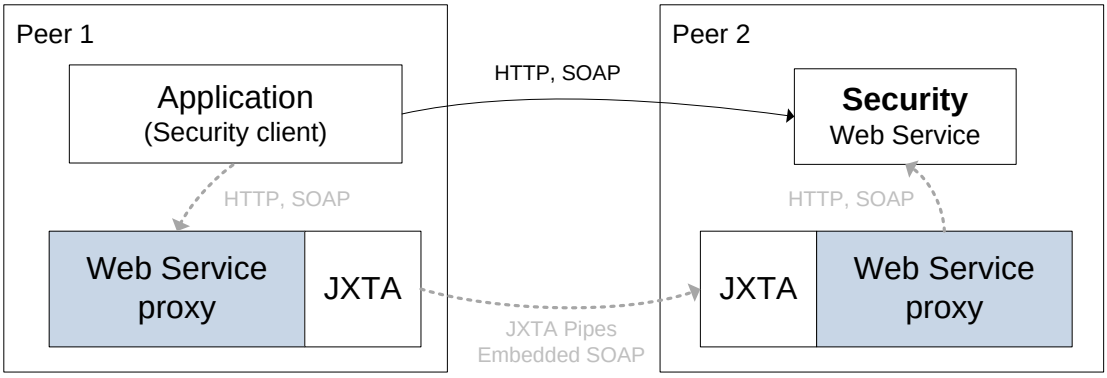


**Figure 4.5 – Direct and relayed Web Service invocation**

The complete process is described as follows:

- The service address in the application's configuration is changed to a fixed location (e.g. http://localhost:8080/wsProxy); this is the actual running endpoint, not the WSDL location. In fact, the proxy is never aware of the WSDL definitions at any point;
- The proxy parses the incoming HTTP SOAP request and retrieves the namespace and the operation name (see example below); no actual service functionality is provided here;
- The proxy, which has a JXTA client, then proceeds to query the network with the operation's unique identifier and in return the Peer 2 JXTA identifier is obtained;
- If required, the SOAP envelope is divided into smaller segments (each message can have up to 64k) and sent in messages to the provider peer;
- The receiving peer acknowledges there is a SOAP request, locates the execution address (either local or remote) and resends it to the Web Service;
- The reverse process is identical, with the SOAP response travelling between the proxies and back to the application.

```
<?xml version="1.0"?>
<soap:Envelope
   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

      <soap:Body xmlns:m="srv://security.infrastructure.dl">
            <m:Hash>
                  <m:input>Generate an hash of this text.</m:input>
                  <m:mode>MD5</m:mode>
            </m:Hash>
      </soap:Body>

</soap:Envelope>
```

With this approach we are able to successfully invoke Web Services through the P2P network. This allows for interoperating with Web Services which could be otherwise unreachable: not only those behind firewalls or NAT systems, but also those only locally available (not even inside a LAN, only at the machine). Other significant result is that it can also allow a public Web Service located at an internet address to be used by a computer without internet connection.

Also, unlike the methodology adopted by JXTA-SOAP, there is a transparent transition from the regular HTTP invocation process to that occurring in the P2P framework. There is no change required in applications other than updating service network addresses. Also of great importance is the fact that the introduced proxies do not limit the frameworks or languages one can use. Both the Web Services and the proxies can be created using any suitable framework or language (Java, .NET, PHP, etc.).

There is one final issue to be addressed, and it relates to how a peer (its proxy) will be aware if it can use the HTTP channel instead of the JXTA one. The simplest case is when the URL in search results starts with “http://localhost”. In those situations, the services will evidently only be available locally, so the service call is proxied through the P2P network. In the other cases, the Web Service proxy should try sending an (empty) HTTP request to the network address. If there is a response (possibly an erroneous SOAP message), the service is within reach. In either case, the performance of this trial-and-error procedure can be improved by caching such information.

### *4.4.3 Replication*

We have discussed in the previous sections how a wide network of services can be made available and discoverable in a P2P network. One extra optimization layer can be set up on top of this service network. Since JXTA has Java and C/C# bindings, we can safely assume most JXTA peers will be running the Java or .NET framework. Let us consider a Java implemented image processing Web Service which only requires a specific minimum framework runtime version and a group of JAR files as its dependencies. To make such service run on a different node, one

would have only to assure those files existed at the target peer (eventually updating the CLASSPATH) and that its runtime version met the requirements. Similarly, replicating a .NET service could require a version of the framework and some DLL files.

It therefore becomes apparent that services could be replicated on a P2P network to increase availability and eventually responsiveness. Such concept does not differ much from file replication, which is implemented by several file-sharing P2P applications. The requirements and dependencies make it however a lesser trivial issue to address. Many services may need more complex dependencies (such as installed programs or libraries) or even have specific hardware requirements.

Describing and managing software (and hardware) dependencies is a difficult task, and several issues and possible conflicts must be taken into account. For now, let us consider the simplest case: a self contained executable or folder with no installation or CLASSPATH modification required. For such components, one could think of replication as yet another service available at some peers (a service “push”), which could be published and discovered as any other. The input parameters of such service are the required resources (executables, WSDL, and dependencies).

On a different level, replication mechanisms can also enhance the peers indexing capabilities. As mentioned in section 4.3.2.1, the capability of indexing a specific file or format is obtained by having a handler. In the case of Java, this is usually nothing more than having a class file (and eventually some file dependencies) and adding an entry to a configuration file.

### *4.4.4 Orchestration*

In the previous sections the issues regarding service discovery, availability and invocation in a distributed P2P network were discussed. While the proposed modifications in the P2P layer can be seen as independent to any specific service environment, one must think in terms of a business process management and execution application to fully take advantage of them.

#### 4.4.4.1 Dynamic binding

The BPEL specification already supports the concept of dynamic partner links. In a static BPEL process, the partner link information is defined at design time. One can, however, declare a generic and abstract service (a template) whose endpoint is set later in the process (for instance in an assign task). Such strategy has the disadvantage of forcing BPEL definitions to be rewritten in order to accommodate such dynamism.

By using the service invocation proposal discussed in section 4.4.2.2, a BPEL engine can take advantage of the available distributed (and eventually replicated) services to dynamically discover and invoke them in a very simple way. The only requirement is that addresses are all replaced by the local proxy address. When invoked the proxy will, in runtime, search service providers in the P2P network. The main advantage of this approach in the scope of business process is related to the inherent dynamism: service providers (peers) can leave and join the network in what can be long-time running processes, hence finding a suitable service only at this time is more appropriate.

There are basically two implementation choices: either change the addresses in runtime in the BPEL engine or replace them on the XML definition document itself before the process starts. Only the later seems a reasonable option, since the former requires developers to change components or modules in the engines, a task that will differ in each BPEL engine one wishes to support.

#### 4.4.4.2 Process delegation

The opportunities created by the service network are not limited to dynamic discovery. Traditionally, BPEL execution is a centralized process, in which service calls are dispatched to partner links and state (the process variables) is centrally managed. However, distributing the orchestration process by the service providers has several advantages, especially in high load scenarios and/or when there is a large amount of data being transferred between service providers and consumers. A careful partitioning process can reduce the number of messages and amount of data transferred and increase throughput.

Previous work assumes all partner nodes have BPEL capabilities [108][104], which may not be convenient in most enterprises, or that the infrastructure of the execution environment is known a priori [110]. We can however fall back to an always working solution.

Lets us consider our initial (starting point) engine is capable of dynamically discovering services. Before the execution starts, the runtime can find not only the service providers but also which nodes offer BPEL execution – since BPEL is seen itself as a Web Service, it can as easily be referenced in the indexes. If no other engine is found, process management must proceed as usual – in a centralized fashion. If, however, one or more engines are found, the BPEL process definition can be partitioned and parts of the process delegated to those peers. If any of those engines are P2P aware, this procedure could eventually be further partitioned.

Without the “BPEL in every peer” assumption, the partitioning mechanism proposed in related work is no longer valid. Nevertheless, some principles remain true: when there is parallel execution (a flow activity), an entire branch can still be partitioned if the first invoke activity exists at a BPEL-capable peer.

Furthermore, information about the services themselves could be used to try to infer the best tasks to be delegated. Process delegation can greatly reduce the amount of data being transferred by eliminating the round trips in the invocation calls. We are therefore interested in those services whose transmitted messages/variables are predictably large, particularly in the response message. While there is no standard way to know a priori which those services are, a few assumptions could be made.

The return type of a service, for instance, can provide hints on the extent or size of the response message. It is safe to assume that the efficiency gain will likely be much smaller when delegating the process to a service returning an integer than the gain when doing so on a service returning an array of bytes. We suggest the enforcement of a simple rule: perform no process delegation if the

next service returns messages with simple types (numeric, Boolean, and strings or complex types based on these types).

By further refining the orchestration mechanism this efficiency and performance gain can be increased:

- By searching peers which provide more two or more services to be executed in sequence, some data transfers can be eliminated;
- Define the notion of “portable” services, transferable between providers without complex dependencies or installation, can be replicated to augment responsiveness;

If one combines the data storage capabilities of P2P with this service overlay there is no need for the final step (sending data to a repository service).

#### *4.4.4.3 A sequential service example*

Let one consider the example of inserting a document such as a thesis in a digital library system. For simplicity of argument, let us assume input data consists of one PDF file and one XML document with the required metadata. The insertion process could consist in the following independent tasks:

- Converting the PDF file into individual image files;
- Creating thumbnails of different dimensions;
- Extracting the text from the PDF;
- Generating an unique identifier for the thesis;
- Storing data and metadata in the repository.

The operation sequence with a BPEL orchestration server is presented in Figure 4.6. When both the PDF and XML files are submitted to the server, the process defined in the BPEL description invokes the services from the providers included in the declaration.

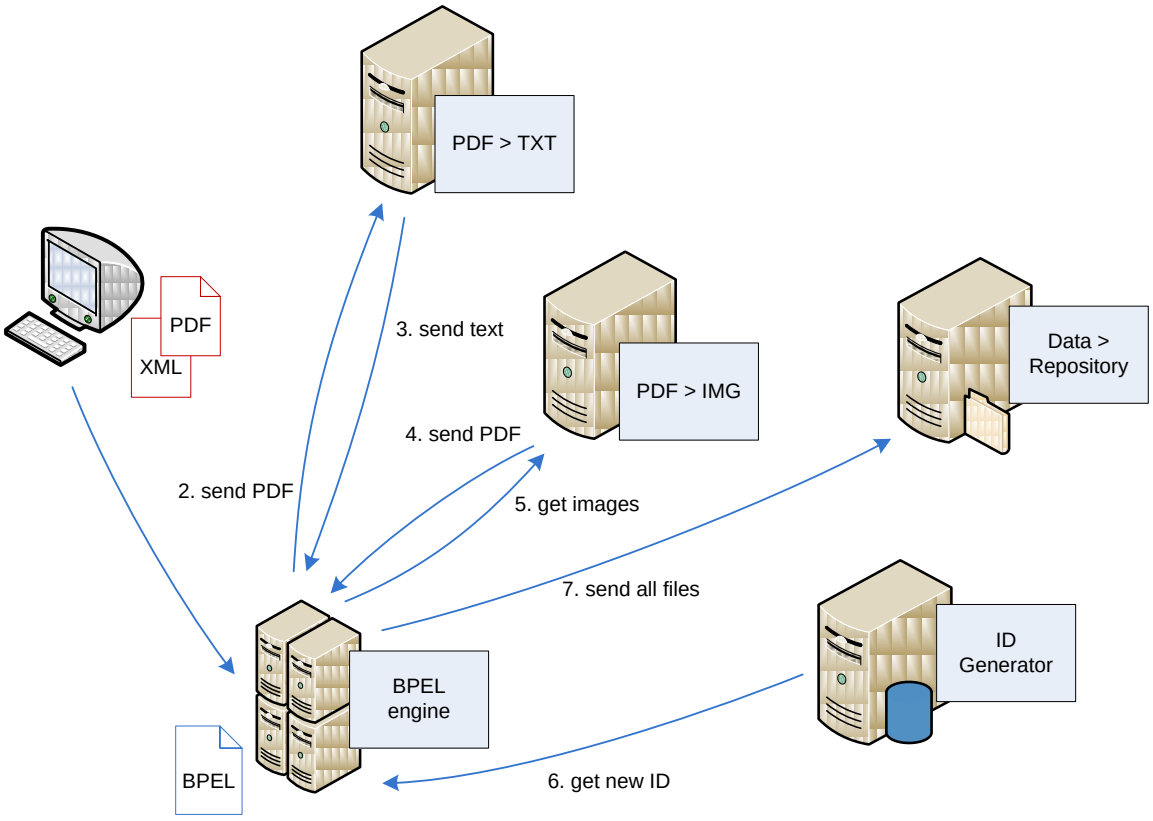


**Figure 4.6 – A common centralized orchestration in a digital library system**

If we discard the network packages of the ID generation message and the XML transmission to the repository, the total size of data transmitted in this orchestration is approximately:

$$S_{trans} = 3 * S_{PDF} + 2 * S_{TXT} + 2 * S_{IMG}$$

where $S_{PDF}$, $S_{TXT}$, and $S_{IMG}$ are the sizes of the PDF file, extracted text, and extracted images, respectively. If these have the values of 60MB, 1MB, and 30MB, for instance, the data transmitted over the wire amounts to over 302 megabytes.

Figure 4.7 depicts a decentralized version of the orchestration for the same digital library process. After the completion of each task, the service provider can either invoke the following service or transfer execution to other node. For example, since the ID generation service does not require the original document to

be inputted, it is better to invoke it to obtain a new identifier rather than transferring the orchestration control.

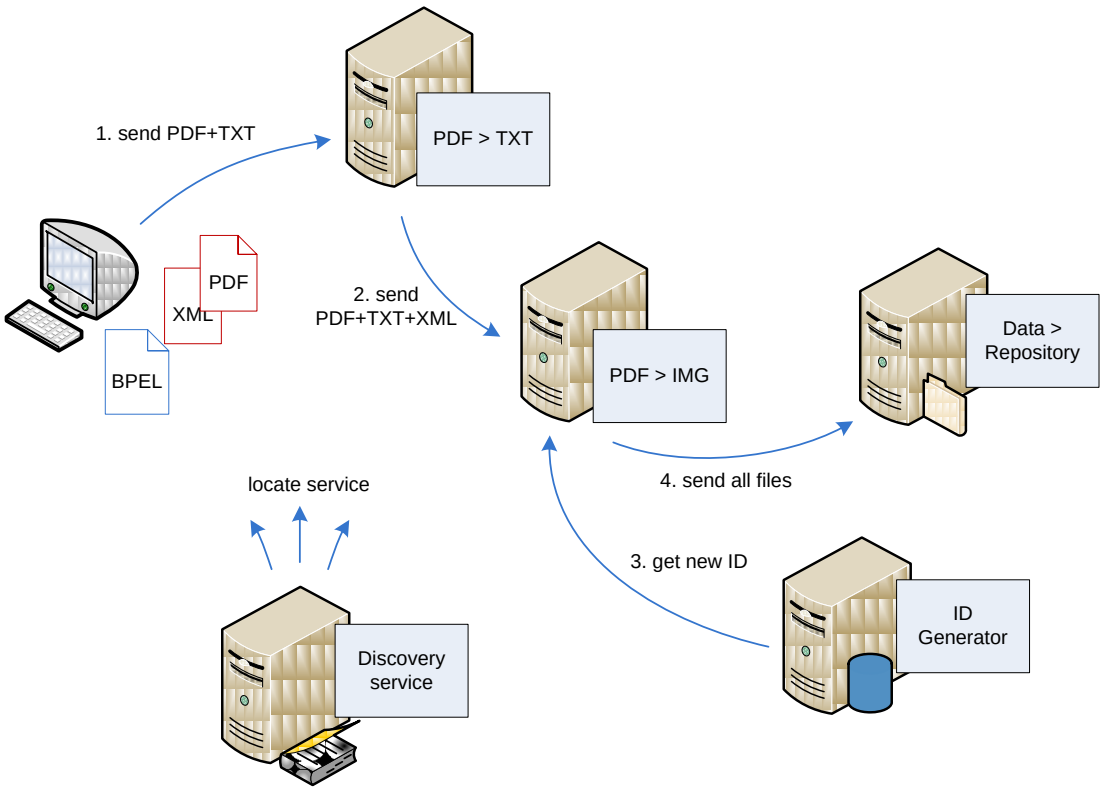


**Figure 4.7 - Decentralizing the orchestration of the digital library process**

With this configuration and with the same considerations used in the centralized scenario, the data transfer size is reduced to about:

$$S_{trans} = 3 * S_{PDF} + S_{TXT} + S_{IMG}$$

which amounts to 212 MB with the sizes from the previous example, a reduction of about 30%. The diagram illustrates another important concept. In a dynamic environment, such as P2P, service providers can enter and abandon the network, and the orchestration should not therefore be tightly bound to specific peers. Instead, services can be dynamically discovered and providers assigned to the

process, either by using a central directory (such as UDDI) or by using other available querying mechanisms.

This simple example demonstrates how properly distributing service orchestration can reduce the amount of data transferred between services. Network usage is however only one of the values we can try to optimize. In a distributed application we are also usually interested in reducing completion time and increase throughput.

Let us consider a real case scenario, in which a digital newsstand website allows registered users to view a range of newspapers as they were published. The website receives PDF files from publishers, which are converted into an image format (JPEG) to be shown in a viewer, and whose texts are extracted for searching purposes.

As part of the submission process, several services are invoked:

- Image conversion/resizing;
- Automatic image whitespace cropping;
- PDF text extraction;
- Optical character recognition (OCR);
- Storage (whose response is the new system identifier) and indexing.

Figure 4.8 depicts a functional diagram of how this process is implemented.

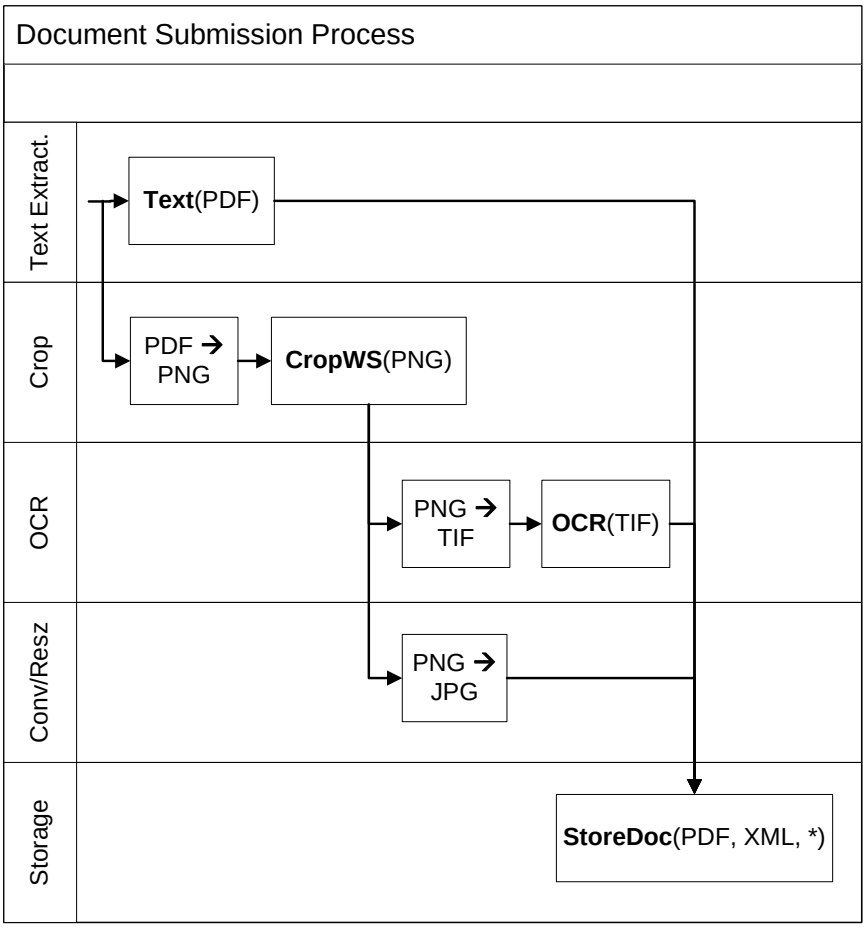


**Figure 4.8 - Cross functional diagram of the document submission process**

The input to this process is a PDF file and a XML document with the metadata. The process starts with two parallel branches. In the first one, the text from the PDF file is extracted. In the second, the PDF is converted to an array of PNG files, whose white space is then cropped. The resulting images are then used to make an OCR (whose service input must be in TIFF files) and to convert to the final, screen resolution, JPEG images. The final activity consists in sending all non-intermediary files to a storage service.

Assuming each of the blocks in the diagram represent a service in a different peer (the worst case scenario), there is a large amount of data being passed back and forth through the wire. With centralized orchestration, one expects the total amount of data to be:

$$T = 3S_{PDF} + 5S_{PNG} + 2S_{TIF} + 2S_{OCR} + 2S_{TXT} + 2S_{JPG} + S_{XML} + S_{ID}$$

where $S_X$ represents the message size of the transmission of X.

The simplest improvement one can do in branched processes is to delegate an entire branch of activities. Let us suppose the image conversion service is available at a BPEL-capable node. In that case, a BPEL process can be made with the activities in the “OCR” band from the diagram. By doing so, the PNG to TIFF conversion call is replaced with a process start call and, since the TIFF files don’t have to be returned to the original caller, those response messages no longer have to be transmitted through the wire. In this particular digital newsstand application, the intermediate TIFFs generated are about 3MB each, and so this modification would reduce close to 120 MB of traffic in a 40 page newspaper.

This procedure could be repeated and, in the optimal scenario where all peers can run BPEL processes, the partitioning algorithm could be identical to those used in the related work. However, some delegation could prove to be counter-productive: consider there were services just before the storage stage dedicated to provide unique identifiers, produce checksums, or calculate hashes based on the metadata of the new document. Delegating the orchestration of one those services and the storage to those providers would actually increase network usage: instead of invoking the first service, receiving the id/checksum and sending all to the StoreDoc, PDF and image files would have to go to the first service and from there to the StoreDoc provider. Therefore, instead of

$$T_{final} = 2S_{XML} + 2S_{ID} + S_{PDF} + S_{JPG}$$

we would have

$$T_{final} = 2S_{XML} + S_{ID} + 2S_{PDF} + 2S_{JPG}$$

which represents one less $S_{ID}$ but one more $S_{PDF}$ and $S_{JPG}$. Since the id/checksum service has a predictably small (numeric) response message, no delegation would take place.

A final optimization could consist in trying to merge activities in peers providing multiple consecutive services. Although this could greatly reduce network traffic, it would be difficult to analyze the improvements of this strategy if factors such as throughput were to be weighed. The case of the last service called (storage) is however a particular one – if the P2P network were to be used also as the storage medium, this service could be directly executed by the caller peer.

#### *4.4.4.4 Implementation*

Unlike the simpler implementation made with the discovery improvement (a SOAP proxy is relatively easy to develop), creating a BPEL process delegation mechanism is a non trivial task.

By following the same dynamic network assumptions, the BPEL decomposition should not be made before process start, and instead before service invocation. Furthermore, a simple proxy is no longer sufficient, since invoking a peer's regular Web Service or its BPEL engine Web Service would require different input data. One has therefore to change the BPEL engines themselves and, without an out-of-the-box solution, that task requires one to perform different implementations for each engine.

#### *4.4.4.5 Tracking progress*

A common feature in process execution solutions is that of showing the current state of the workflow to the user (process monitoring), either in a web page available at some port or within a desktop application. While keeping track of this progress is simple in a centralized scenario, since the engine has all the information of the activities currently executing and of those already finished, doing so in a decentralized orchestration environment is not as trivial.

While this may be a non-critical issue and one which only occurs for those engines enhanced to support BPEL delegation, there are nevertheless a few possible approaches to handle it. One can simply ignore the existence of composite services (processes) within the process. In this always-working solution, the new process is seen by the end user as another basic activity.

The original BPEL engine can also provide links (if any) to the feedback page on the other engines. This has a number of issues related with the underlying P2P network. While a service in the first engine was successfully invoked, there is no guarantee that the end user can access the other inner process. Network topology, security issues, and local/private services are some of the possible factors that can prevent these two entities from connecting.

Another possible approach would be to embed the status response from the inner process into the first engine's feedback page (or eventually use some sort feed if available). This would require however the address of such pages to be known a priori by the calling engine.

## 4.5 RESULTS

### 4.5.1 Search engines evaluation

The role of a search engine in a digital library and in particular in hybrid scenarios has been discussed in section 4.3.2 and previous chapters. With the goal of finding a suitable engine for our new architecture, one that could be properly integrated into our infrastructure, an evaluation of six free or open-source available engines was conducted [132]. Having considered the requirements of a hybrid P2P infrastructure for digital libraries, the engines were analyzed and compared primarily focusing on six characteristics:

- **XML indexing**. By default, typical search engines index text files, HTML, PDF and eventually office documents. However, as XML becomes the standard de facto for description storage and transmission, and it is already ubiquitously used, modern search engines should provide means for searching XML documents.
- **Ability to move and merge indexes**. In a P2P based Digital Library, search engines will be running on each peer, indexing the local node repository. Periodically – or whenever an update is made –, the local indexes must be sent to the super-peer where they are merged. This requires indexes to be movable between nodes.

- **Platform independence**. Ideally, the P2P network is able to support different operating systems, so components should be interoperable. The search engines should have different APIs while keeping generated indexes transparent to the programming language.
- **Ranking**. Effectively ranking results is one of the most important features of search engines, since users tend to view only the first result pages [133].
- **Off-line searching**. In some engines, the index and search services are not clearly separated, so performing a query in the later requires the first to be running. This behavior is not recommended primarily for two reasons: if there is critical or private data being transferred between nodes, the system may enforce some sort of data encryption to assure it is not tampered with. If the data is to be indexed, however, it is necessary to use a procedure that decrypts indexes and encrypts the data again before saving it on a node's hard drive. Also, such tightly coupled indexing and searching mechanism makes it difficult to search indexes stored in a different computer.

The engines evaluated in the test were Indri [134], Apache Lucene [135], Microsoft Indexing, Swish-e [119], Terrier [136], and Zebra [137].

The results of the evaluation are summarized in Table 4-2 (with latest release dates updated) and Table 4-3. The performance benchmark was made in two stages: the first with a repository comprising of 25.000 plain text files and 5.000 XML files from the SInBAD's digital repository and the later with extra 25.000 miscellaneous and HTML files. Speed measurements were all taken from the same machine – a Pentium 4 with 3.2 GHz, 2 GB of RAM, and a SATA 7.200 rpm hard drive running Windows XP.

**Table 4-2 – Indexing engines feature comparison**

| | IN | IS | LU | SW | TE | ZE |
|---|---|---|---|---|---|---|
| **Incremental indexing** | ✓ | ✓ | ✓ | | | ✓ |

| | | | | | | |
|---|---|---|---|---|---|---|
| **XML support** | ✓ | filter req. | filter req. | ✓ | ✓ | ✓ |
| **XML native namespaces support** | | n/a | n/a | ✓ | | ✓ |
| **Offline searches** | ✓ | | ✓ | ✓ | ✓ | ✓ |
| **Platform independent indexes** | ✓ | | ✓ | ✓ | ✓ | ✓ |
| **Latest stable release** | Dec. 2009 | 2003 | Nov. 2009 | Apr. 2009 | Jan. 2009 | Nov. 2009 |

From our analysis, we consider Lucene to be the search engine to use on our P2P digital library. Lucene has the best searching performance, is platform independent, supports incremental indexes, is a fast evolving application and, although it does not support XML natively, parsers can be easily constructed. It is also available in several APIs, such as C, Java, and .Net. Swish-e provides a wide range of options and it also has an excellent performance, but it lacks more powerful APIs and the ability to use incremental indexes.

**Table 4-3 – Indexing engines performance comparison**

| | **IN** | **IS** | **LU** | **SW** | **TE** | **ZE** |
|---|---|---|---|---|---|---|
| Stage 1 | | | | | | |
| **Index size (MB)** | 88 | 41 | 74 | 27 | 68 | 157 |
| **Full index (s)** | 6854 | 300 | 3478 | 38 | 228 | 59 |
| **Full-text search (s)** | 21.4 | 44.6 | 8.5 | 38.5 | 34.0 | 59.2 |
| **XML search (ms)** | 15.6 | 14.8 | 1.9 | 46.6 | 15.7 | 58.1 |
| Stage 2 | | | | | | |
| **Index size (MB)** | 223 | 104 | 325 | 83 | n/a | 1090 |
| **Full index (s)** | 72313 | 604 | 11280 | 226 | n/a | 495 |
| **Full-text search (s)** | 27.3 | 46.5 | 10.9 | 61.2 | n/a | 69.8 |

### 4.5.2 Adapting SInBAD to the new architecture

The preliminary results from SInBAD serve as the building blocks for the work and analysis that followed. The proposals made in this chapter follow the discussion on the strengths and weaknesses of SInBAD and allow for the proposal of a new architecture for the system.

We will start by categorizing the existing services according to the service taxonomy discussed in section 4.4.1.4. As shown, Business services include the "abstract" services (i.e. related to conceptual notions such as a thesis or poster instead of the actual physical representations), Enterprise services include the user and role based operations (which interoperate with RCU), and Application services include all the "helper" functions needed under the hood for the different catalogs.

Infrastructure services will contain functionality typically available at any node. The Storage and Indexing subcategory groups core and mandatory services (I/O, Service interoperation, Search), Security includes services related to security operations (hashing, encryption) and Document Processing services deal with resource conversion and transformation (thus acting as a distributed replacement for Caterpillar).

**Table 4-4 – Service taxonomy for SInBAD**

| Category | Service examples |
|---|---|
| **Business** | InsertThesis, InsertPoster, …<br>UpdateThesis, UpdatePoster, …<br>GetThesis, GetPoster, …<br>SearchTheses, SearchPosters, … |
| **Enterprise** | AuthenticateSinbadUser, GetUserRoles, AuthorizeOperation, … |
| **Application** | CheckMagazinePdfExists, MoveVideoFile, GetPublicationPageCount, GetThesisUrlFromId, MapMetadata, CreateCatalog… |
| **Infrastructure** | GenerateId, CreateIdScope, RemoveIdScope, … |
| **Infrastructure** | Store, Get, Search, Reindex, InvokeService… |

| | |
|---|---|
| **> Storage & Indexing** | |
| **Infrastructure > Security** | Hash, SymmetricEncrypt, SymmetricDecrypt, AsymetricEncrypt, AsymmetricDecrypt… |
| **Infrastructure > Document processing** | ConvertImage, ResizeImage, ExtractText, OCR, … |

After the proper analysis and categorization of required services, one should design BPEL definitions for every composite service, i.e. those who rely on other services. Infrastructure services should be autonomous and therefore should not require a workflow definition.

The proposed system architecture is presented in Figure 4.9. As can be noticed the diagram is very similar to Figure 4.1 since only the applications layer is replaced by the SInBAD System; the “infrastructure” models, P2P and Service, remain unaltered. The SInBAD system is composed by front-end applications (Website, Web Services, and OAI-PMH provider), a module with the system’s core logic, a BPEL engine, and a synchronization module (SInBAD Sync). The system is configured by a set of configuration and service description files.

External access to SInBAD resources is accomplished by using one of three interfaces: 1) the Website is the primary front-end for generic users to search and access resources; special users can also login and use the back-office to create or update data; 2) Web Services allow external systems such as e-ABC to interoperate with SInBAD; 3) the OAI-PMH module provides metadata in a standard compliant format so that open archive harvesters can index it.

Each of the theses interfaces uses the DL Logic module to search and access data from the underlying network. This module is the “glue” responsible for implementing the core functionality of the system. Aside from assuring communication with the network (using the P2P and Service modules) it coordinates two other components: a BPEL engine and a synchronization module.

By transposing the (previously hardcoded) business processes to BPEL descriptions, the BPEL engine can bring to the application the benefits of dynamic

service discovery and binding. Also, if other BPEL engines exist in the network, process partitioning can be employed. This is accomplished with the interaction between BPEL-e and the Service module (namely the Service Proxy).

SInBAD Sync is responsible for the synchronization of the application configuration and rules between the nodes and to keep business processes in existing backup peers (B-Peers) updated. If the node where the SInBAD application is shut down or unexpectedly leaves the network, a B-Peer can assume its role and eventually continue providing all front-ends.

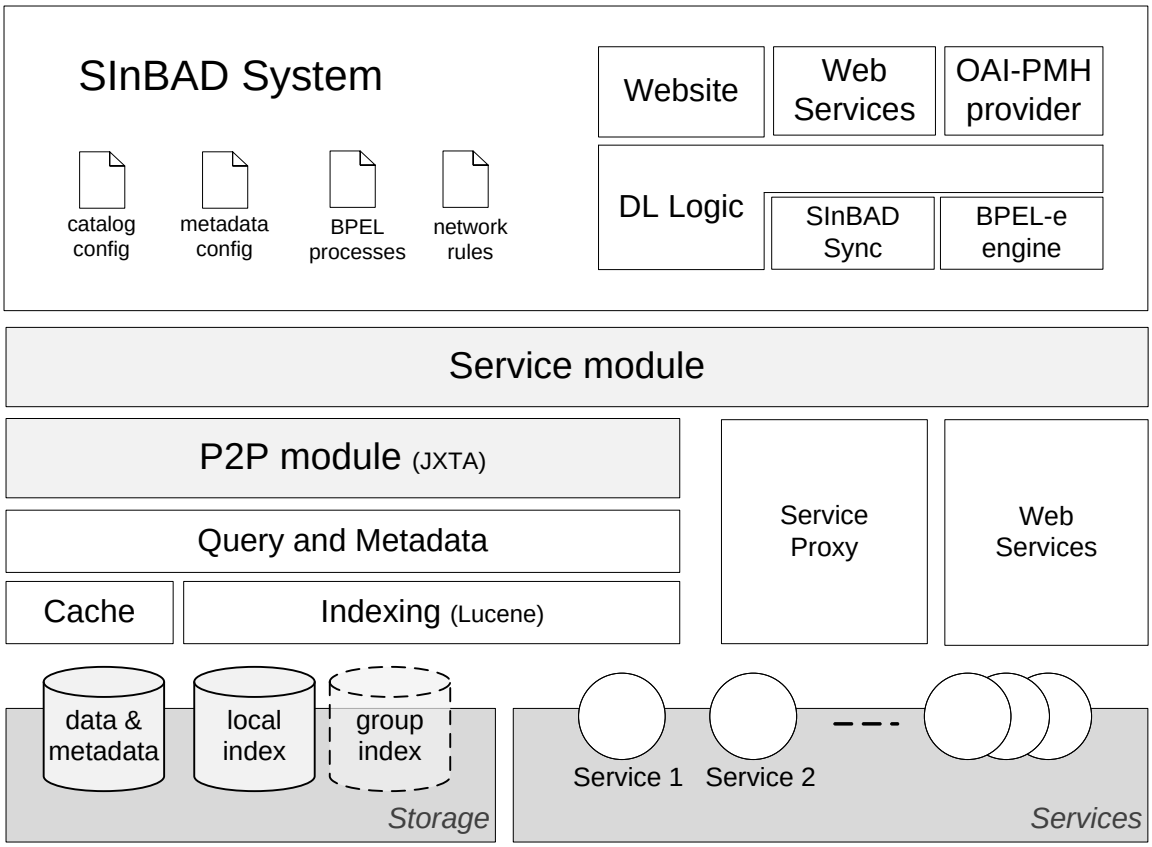


**Figure 4.9 – Proposed SInBAD system architecture**

Regarding the P2P layer, the Indexing module should use an engine having in consideration the features discussed in 4.3.2, such as movable and mergeable indexes (Lucene seems the best candidate at the moment). Also a considerable

difference from the architecture currently in use concerns to the concept of resource, which aggregates both files and services. All the querying, storage, and indexing modules have therefore the abstraction notion of a resource which has specific metadata to be indexed and searched for.

## 4.6 SUMMARY

As a sequence of the analysis of the strengths and issues of the SInBAD system, made in the end of the previous chapter, we have presented a novel architecture based on P2P and services.

Regarding the P2P infrastructure, we have started by acknowledging the need for an indexing and search engine and performed a comparison analysis between available free or open-source tools. With the results of that test, an indexing configuration was devised.

We then introduced the service layer in the P2P network, and presented a proposal that improves service discovery and widely broadens the range of possibilities for service invocation. The advantages that can be derived from the presented approach are also applied to the execution of business processes.

A generic resource concept – applicable to both documents and services – was also proposed as a main direction in the system design. This uniformization allows for generic rules to be applied for both cases in a variety of situations including metadata definition, indexing, and searching.

# CHAPTER 5 – Conclusions

This thesis has presented a novel architecture for digital libraries based on peer-to-peer networks and service oriented technologies and concepts.

This chapter summarizes the results and contributions realized in this doctoral work in its three main stages: the study of DLMS related technologies, the development of the University of Aveiro digital library and archive, and the conception an architecture proposal for digital libraries based on P2P and SOA.

In the first part of this doctoral dissertation, we discussed the rationale which led to the conception of the current architecture of SInBAD, the digital library system of the University of Aveiro. We started by examining popular DLMS under the light of research advances and recommendations given by workgroups from a known best practice network for the excellence of digital libraries. We concluded they lacked important features or where of limited use in decentralized scenario.

We then analyzed the main characteristics of P2P networks and what are its main advantages and issues in the scope of a digital library infrastructure. In conclusion of the analysis made we have selected hybrid P2P networks with unstructured data as the ideal configuration for digital libraries.

We concluded this preliminary work by benchmarking search performance with and without a super-peer in a small LAN network. With such an experience, we thus validated that a hybrid topology is also suitable for very small networks.

The result of this work's second stage is the conceived architecture and the SInBAD system itself, which successfully overcomes limitations found in existing digital library management systems, such as 1) lack of restrictions in the access to copyrighted documents, 2) use of a centralized data repository, 3) rigid description model, or 4) limited search capabilities. It is a service oriented application which can store resources in a distributed manner and handle (search, view, manage) heterogeneous metadata from different catalogs in a flexible way.

The designed and implemented University of Aveiro's digital archive and library is the primary result of this work. Together with e-ABC, the system became the university's institutional repository. The implementation issues and the emerging challenges were thoroughly analyzed in CHAPTER 3. In order to describe very heterogeneous data we have analyzed different metadata standards and created schemas for the different resources using a Dublin Core base and terms from other specific standards.

The conceived SInBAD architecture was based in the concept of subsystems – network nodes with coherent and cooperating microsites exposing data using Web Services with common predefined interfaces and specific methods related to the resources in scope. On top of the subsystems, the SInBAD portal is the main entry point for system, and is responsible for the transparent interoperation between subsystems. Apart from implementing a single sign on and presenting institutional information, the portal's main feature is related to the repository wide searches. It can aggregate information and search results from the subsystems

and present it to users in a uniform way. This capability is also used to feed the created OAI-PMH provider interface.

An important part of that system – the one which enables the system to handle distributed resources – is however hidden from users: DisQS. The DisQS system was the first approach to implement a distributed storage and service mechanism.

The system is composed by a number of Agents which are coordinated by a Manager, and it allows resources with custom metadata models to be stored, indexed, and searched for. It can also be configured with specific settings per catalog, therefore allowing for distinct rules to be applied for different content.

DisQS is also service oriented, and all communication between an Agent and the Manager – search requests, storage and retrieval, etc. – is made through Web Services interfaces.

What also makes DisQS different from common distribution of resources – focused on data only – is its ability to distribute the workload of applications using it. With DisQS, instead of preparing and transforming data before storing it, the Manager can delegate most of those tasks to the Agents where data will be stored.

By the end of 2009, the system stored over 2.500 thesis and dissertations, 300 magazine articles, 6.500 digitized posters, 2800 photographs, 6.700 jazz records, and 600 jazz books, among others.

Almost every resource available in the repository can be accessed by any user (regardless of whether he has or not a connection to the university) from outside campus. The exception to the rule is copyrighted content which must have a controlled access, such as books and music records. The most popular content includes the doctoral theses and the master dissertations.

The January of 2010 access reports show about 14.000 visits and 82.000 page views. From those visits, about 2.000 are from outside Portugal (mostly from

Brazil, and less significantly from France, Cape Verde, Spain, and USA) and 10.000 from outside Aveiro.

Its open archive infrastructure (OAI-PMH) allows metadata from the repository to be harvested by external services such as RCAAP – Repositório Científico de Acesso Aberto em Portugal [138].

The final stage of this work results from the analysis of the designed digital library and the issues which were identified. More importantly, it focuses on how the opportunities offered by recent computational models, such as peer-to-peer and service oriented architectures, can greatly improve performance, robustness, and flexibility.

CHAPTER 4 presents a novel digital library architecture based on P2P and SOA which overcomes those issues. The architecture addresses 3 main areas:

- **P2P for a digital library**. Traditional P2P applications handle only very simple metadata which make them inadequate for digital libraries. We have shown how an existing P2P framework can be adapted and integrated with an open-source search engine to successfully index and search very heterogeneous metadata in a flexible way. We also presented the concept of B-Peer, to increase the network robustness and availability.
- **Service publishing and discovery**. We have shown how a service oriented application can make use of a P2P infrastructure to dynamically find matching service providers. A service taxonomy was also proposed to help software developers easily find suitable services. Finally, a novel approach was conceived to seemingly handle both data and services as abstract services, which can be described, indexed and queried using a common metadata schema.
- **Service invocation**. This work has presented a mechanism which allows two computers to interoperate using Web Services in scenarios with very limited or no connectivity. We finally discussed service orchestration and the adaptation of business process to take

advantage of the underlying service network, by resorting to dynamic service binding and delegation of sub processes. This work has shown how one can achieve a higher performance using proper orchestration of services available in a P2P network.

Even without having developed a fully functional digital library using this architecture, we have shown how the combination of P2P and SOA help overcome limitations in the current state of the art and can offer a novel resource handling paradigm in a distributed scenario.

Since the importance of the role of a search engine was clearly discussed throughout this dissertation, an evaluation of a set of free and open-source search engines was made in the context of the proposed architecture.

We finalized our work by showing how the current SInBAD infrastructure could be adapted to this new architecture, which includes but is not limited to: adopting the service taxonomy, using a BPEL engine for business processes, creating a synchronization mechanism to maintain a backup peer up to date, and changing the metadata and indexing mechanism.

## *5.1 FUTURE WORK*

Although prototypes were developed as proof of concept for parts in this proposal, as a future work we wish to create a fully functional running environment. One wishes to fully adapt the SInBAD digital library and archive system to the new architecture in order to better evaluate and validate the contributions made in this doctoral work. The orchestration decentralizing process is predictably the most complex task, since it will require modifying or developing new modules of a given BPEL engine.

Several other investigation areas remain open for further research and more comprehensive analysis. Regarding the search infrastructure in the P2P network, it is important to not only assure determinist results but also guarantee a proper ranking mechanism. When querying for distributed (and replicated) resources, rank values are commonly generated with repository dependent formulas making

identical resources to be ranked very differently according to the size and content of the local collections where are located. Having a unique centralized index solves the issue only to some extent since other problems arise from the centralization such as lower fault resilience and higher indexing network traffic. A simple query language was presented for prototyping and testing purposes. A more comprehensive study of query languages should be made in the future in order to search resources in a standard fashion.

Regarding data stored with the P2P infrastructure, an important research area for the future is the study of mechanisms to automatically replicate resources within the network. This includes the replication of metadata, indexes, catalog information, and data itself. Although replication can be especially important in a digital library to assure a higher availability, it does increase network traffic and creates a versioning problem.

Also regarding replication of resources, we briefly approached the problems that arise when trying to replicate a service in a distributed P2P based infrastructure. Several issues are subject of research in other workgroups, from deployment itself, replication scheduling and priority management, dependencies handling, and routing.

The author has also participated in a workgroup regarding the application of grid computing in digital libraries, which is research topic closely related to P2P. More specifically, the workgroup developed and evaluated prototypes to more efficiently execute CPU intensive tasks of digital libraries with Grid computing [38]. Unlike the research made with P2P, the goal we tried to achieve with grid computing was to optimize a single and complex service, by subdividing it into smaller and distributable tasks.

Although significant performance gains were achieved, services had to be developed or adapted in a way that made it possible to send it to “executors” when they were available. In the future, we hope to further investigate this research area and try to combine the grid performance with the P2P flexibility.

Some efforts of the workgroup have also been applied to analyzing data preservation mechanisms using grid computing. Although a prototype was already developed using a rule-based grid platform and the SInBAD OAI-PMH and Web Services interfaces [29], we wish to work further on this topic.

The semantic web is yet another topic which has been actively discussed and researched in the last years. We believe a semantic layer should be added on top of the architecture we have designed, closely integrated with the metadata and indexing modules, to offer digital library applications a higher degree of knowledge of concepts and relationships.

Finally, security is the one of the obvious topics to handle next. This work has deliberately left out of scope security issues which should now be integrated on top of the designed architecture, both at the service and application level. This primarily includes authentication and authorization processes, which should take into account the distributed nature of the system and the resources. Therefore, security methods such as single-sign-on should be taken into account. Also, one should consider enforcing encryption mechanisms in order to prevent unauthorized users from accessing or modifying data.

# References

[1] Britannica Encyclopædia. (2008) Britannica Online Encyclopedia. [Online]. http://www.britannica.com/EBchecked/topic/1017484/search-engine

[2] Digital Library Federation. (2004) Digital Library Federation. [Online]. http://www.diglib.org/about/dldefinition.htm

[3] Barry M. Leiner. (1998) DLib. [Online]. http://www.dlib.org/metrics/public/papers/dig-lib-scope.html

[4] Mary E. Brown. (n.a.) Southern Connecticut State University. [Online]. http://www.southernct.edu/~brownm/dl_history.html

[5] University of California. (n.a.) UC Southern Regional Library Facility. [Online]. http://www.srlf.ucla.edu/exhibit/text/BriefHistory.htm

[6] Vannevar Bush, "As we may think," *Atlantic Monthly*, vol. 176, no. 1, pp. 101-108, 1945.

[7] University of Missouri. (2003) Interactive Digital Library Resources Information System. [Online]. http://www.coe.missouri.edu/DL/iDLR/viewpaper.php?pid=21

[8] Tim Berners-Lee. (1989) CERN. [Online]. http://www.w3.org/History/1989/proposal.html

[9] Robert Kahn and Robert Wilensky. (1995) Corporation for National Research Initiatives. [Online]. http://www.cnri.reston.va.us/home/cstr/arch/k-w.html

[10] William Y. Arms, Christophe Blanchi, and Edward A. Overly, "An architecture for information in digital libraries," *D-Lib Magazine*, February 1997, http://www.dlib.org/dlib/february97/cnri/02arms1.html.

[11] B. Kahle, R. Prelinger, and M. E. Jackson, "Public access to digital material," *D-Lib Magazine*, 2001.

[12] Internet Systems Consortium. (2009, July) Internet Systems Consortium. [Online]. https://www.isc.org/solutions/survey/history

[13] Open Archives Initiative. (2004) Open Archives Initiative. [Online]. http://www.openarchives.org/OAI/openarchivesprotocol.html

[14] IJ Taylor, *From P2P to Web Services and Grids: peers in a client/server world*. London: Springer-Verlag, 2005.

[15] The Gridbus Project. (2008) Grid Computing Info Centre (GRID Infoware). [Online]. http://www.gridcomputing.com/

[16] Webopedia. (2006) Webopedia.com. [Online]. http://itmanagement.webopedia.com/TERM/G/grid_computing.html

[17] Mark Baker, Amy Apon, Clayton Ferner, and Jeff Brown, "Emerging Grid Standards," *IEEE Computer*, vol. 38, no. 4, pp. 43-50, April 2005.

[18] D. Talia and P. Trunfio, "Toward a synergy between P2P and Grids," *IEEE Internet Computing*, vol. 7, no. 4, 2003.

[19] M. Cannataro and D. Talia, "Semantics and knowledge Grids: building the next-generation Grid," *IEEE Intelligent Systems*, vol. 19, no. 1, pp. 56-63, 2004.

[20] OASIS. (2006) OASIS. [Online]. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

[21] Eric Newcomer and Greg Lomow, *Understanding SOA with Web Services*.: Addison-Wesley, 2005.

[22] IBM. (2008) IBM. [Online]. http://www-306.ibm.com/software/solutions/soa/

[23] Carl Simon. (2005) Carl's Consulting Adventures. [Online]. http://carlaugustsimon.blogspot.com/2005_09_01_archive.html

[24] Thomas Erl, *Service-oriented architecture: concepts, technology, and design*.: Prentice Hall, 2005.

[25] Marco Fernandes, Pedro Almeida, Joaquim A. Martins, and Joaquim S. Pinto, "A digital library framework for the University of Aveiro," in *Communicating Mathematics in the digital era*, E. M. Rocha and J. F. Rodrigues, Eds.: A K Peters, Ltd., 2008.

[26] Pedro Almeida, Marco Fernandes, Miguel Alho, Joaquim Arnaldo Martins, and Joaquim Sousa Pinto, "SInBAD - A Digital Library to Aggregate Multimedia Documents," in *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, Guadeloupe, French Caribbean, 2006, p. 173.

[27] Pedro Almeida, Marco Fernandes, Miguel Alho, Joaquim Arnaldo Martins, and Joaquim Sousa Pinto, "SInBAD - Sistema Integrado de Biblioteca e Arquivo Digital," in *XATA 2006 : XML - Aplicações e Tecnologias Associadas*, Portalegre, Portugal, 2006, pp. 139-149.

[28] Pedro Almeida, Marco Fernandes, Miguel Alho, Joaquim Arnaldo Martins, and Joaquim Sousa Pinto, "SInBAD - Sistema integrado de arquivo e biblioteca digital," in *IADIS Ibero-Americana: WWW/Internet 2005*, Lisbon, 2005, pp. 129-136.

[29] Marco Pereira, Marco Fernandes, Joaquim Arnaldo Martins, and Joaquim Sousa Pinto, "SInBAD digital library preservation using IRODS data grid," in *ICSOFT 2009: 4th International Conference on Software and Data Technologies Abstracts*, vol. 2, Sofia, 2009, pp. 107-112.

[30] Marco Fernandes, Joaquim Arnaldo Martins, Joaquim Sousa Pinto, Pedro Almeida, and Helder Zagalo, "DisQS - Web Services based distributed query system," in *XATA 2005 : XML - Aplicações e Tecnologias Associadas*, Braga, Portugal, 2005, pp. 370-371.

[31] Marco Fernandes, Pedro Almeida, Helder Zagalo, Joaquim A. Martins, and Joaquim S. Pinto, "Sistema de pesquisa distribuído (DisQS) para suporte a bibliotecas e arquivos digitais baseado em web services," in *CLME' 2005-ICEM : 4º Congresso Luso-Moçambicano de Engenharia - 1º Congresso de Engenharia de Moçambique*, Maputo, Mozambique, 2005, pp. 1179-1188.

[32] Pedro Almeida, Marco Fernandes, Joaquim Arnaldo Martins, Joaquim Sousa Pinto, and Helder Zagalo, "DisQS - Distributed Query System based on Web Services for digital libraries ," in *ITA' 05 - First International Conference on Internet Technologies and Applications*, Wrexham, Wales, 2005.

[33] Marco Fernandes, Joaquim A. Martins, and Joaquim S. Pinto, "SOPPA - Service Oriented P2P Framework for Digital Libraries," in *Joint Conference on Digital Libraries [doctoral consortium]*, Vancouver, 2007.

[34] Marco Fernandes, Pedro Almeida, Joaquim A. Martins, and Joaquim S. Pinto, "SOPPA: Service oriented P2P framework for digital libraries," in *ICEIS 2008: 10th International Conference on Enterprise Information Systems*, Barcelona, 2008, pp. 215-219.

[35] Pedro Almeida, Marco Fernandes, Joaquim Arnaldo Martins, and Joaquim Sousa Pinto, "Resource aggregation in digital libraries: static vs. dynamic protocols," in *ICEIS 2008: International Conference on Enterprise Information Systems*, Barcelona, Spain, 2008, pp. 405-412.

[36] Pedro Almeida, Marco Fernandes, Helder Zagalo, Joaquim A. Martins, and Joaquim S. Pinto, "Visão geral de sistemas de integração de fontes de informação heterogéneas," in *CLEM' 05 - ICEM: 4º Congresso Luso-Moçambicano de Engenharia - 1º Congresso de Engenharia de Moçambique*, Maputo, 2005, pp. 1169-1178.

[37] Marco Pereira, Marco Fernandes, Joaquim Martins, and Joaquim Pinto, "Service oriented P2P networks for digital libraries, based on JXTA," in *4th International Conference on Software and Data Technologies*, Sofia, Bulgaria, 2009.

[38] Marco Fernandes, Pedro Almeida, Joaquim A. Martins, and Joaquim S. Pinto, "Improving performance of background jobs in digital libraries using Grid computing," in *ICEIS 2008: 10th International Conference on Enterprise Information Systems*, Barcelona, 2008, pp. 221-225.

[39] Sara Silva, Marco Fernandes, Pedro Almeida, Joaquim A. Martins, and Joaquim S. Pinto, "SWIT: An open-source web-based database management system with indexing engine integration," in *EATIS 2008: Euro American Conference on Telematics and information Systems*, Aracaju, 2008.

[40] Hans Schek and Can Türker. (2004, October) DELOS Network of Excellence on Digital Libraries. [Online]. http://delos-old.isti.cnr.it/newsletter/issue2/feature1/

[41] MacKenzie Smith et al., "DSpace - an open source dynamic digital repository," *D-Lib Magazine*, January 2003.

[42] Dublin Core Metadata Initiative. (2008) DCMI. [Online]. http://dublincore.org

[43] William Nixon, "DAEDALUS: Initial experiences with EPrints and DSpace at the University of Glasgow," *Ariadne Magazine*, no. 37, October 2003.

[44] EPrints. (2008) EPrints. [Online]. http://www.eprints.org

[45] Carl Lagoze, Sandy Payette, Edwin Shin, and Chris Wilper, "Fedora: an architecture for complex objects and their relationships," *International Journal on Digital Libraries*, vol. 6, no. 2, pp. 124-138, 2006.

[46] Sheridan Libraries. (2008) Sheridan Libraries. [Online]. https://wiki.library.jhu.edu/display/RepoAnalysis/

[47] Ian H. Witten, Rodger J. Mcnab, Stefan J. Boddie, and David Bainbridge, "Greenstone: a comprehensive open-source digital library software system," in *Proc. ACM DL*, 2000, pp. 113-121.

[48] Carlo Meghini and Thomas Risse, "BRICKS: A digital library management system for cultural heritage," *ERCIM News*, no. 61, pp. 54-55, April 2005.

[49] Y Ma and R.S. Aygun, "Peer-to-peer based multimedia digital library," in *Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory*, 2005, pp. 130- 134.

[50] Carlo Mastroianni, Domenico Talia, and Oreste Verta, "A super-peer model for resource discovery services in large-scale Grids," *Future Generation Computer Systems*, vol. 21, no. 8, pp. 1235-1248, October 2005.

[51] J. Mischke and B. Stiller, "A methodology for the design of distributed search in P2P middleware," *IEEE Network*, vol. 18, no. 1, pp. 30-37, Jan/Feb 2004.

[52] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, San Diego, California, 2001, pp. 149-160.

[53] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, "A scalable content-addressable network," in *Proceedings of the ACM SIGCOMM*, San Diego, California, 2001, pp. 161-172.

[54] Wolfgang Müller, Martin Eisenhardt, and Andreas Henrich, "Scalable summary based retrieval in P2P networks," in *ACM Conference on Information and Knowledge Management*, Bremen, Germany, 2005, pp. 586-593.

[55] The Gnutella Developer Forum. (2003) Gnutella - A Protocol for a Revolution. [Online]. http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[56] Matei Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Proceedings of the First International Conference on Peer-to-Peer Computing*, Linkoping, Sweden, 2001, pp. 99-100.

[57] G2DN. (2007) Gnutella2 Developer Network. [Online]. http://g2.trillinux.org

[58] BitTorrent, Inc. (2008) BitTorrent. [Online]. http://www.bittorrent.com/

[59] Wikipedia. (2008) Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/BitTorrent_(protocol)

[60] Wikipedia. (2008) Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/Napster

[61] Wikipedia. (2008) Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/FastTrack

[62] John R. Douceur, Atul Adya, Josh Benaloh, William J. Bolosky, and Gideon Yuval, "A Secure Directory Service based on Exclusive Encryption," in *18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, USA, 2002.

[63] William J. Bolosky, John R. Douceur, and Jon Howell, "The Farsite project: a retrospective," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 2, pp. 17-26, 2007.

[64] Karl Aberer et al., "P-Grid: a self-organizing structured P2P system," *ACM SIGMOD Record*, vol. 32, no. 3, 2003.

[65] J. Walkerdine and P. Rayson, "P2P-4-DL: digital library over peer-to-peer," in *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, 2004, pp. 264-265.

[66] W. et al. Nedjl, "EDUTELLA: a P2P networking infrastructure based on RDF," in *Proceedings of the 11th international conference on World Wide Web*, 2002, pp. 604-615.

[67] Stratis D. Viglas, Theodore Dalamagas, Vassilis Christophides, Timos Sellis, and Aggeliki Dimitrou. School of Electrical and Computer Engineering. [Online]. http://milos.dblab.ece.ntua.gr/p2pdl/

[68] Old Dominion University Digital Library Group. Old Dominion University. [Online]. http://p2pdl.cs.odu.edu/

[69] Pawel Gruszczynski, Cezary Mazurek, Stanislaw Osinski, Andrzej Swedrzynski, and Sebastian Szuber, "dLibra - Content management for digital libraries," in *Euromedia'2002 - 7th Annual Scientific Conference*, 2002, pp. 28-32.

[70] CollabNet, Inc. (2008) JXTA. [Online]. http://www.jxta.org

[71] Jabber, Inc. (2008) Jabber.org. [Online]. http://www.jabber.org/

[72] Joost N.V. (2008) Joost - Free online TV. [Online]. http://www.joost.com

[73] Krawler Networks. (2008) Krawler[x]. [Online]. http://www.krawlerx.com/

[74] Björn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins, "Peer-to-Peer support for massively multiplayer games," in *INFOCOM*, 2004, p. 107.

[75] Christoph Neumann, Nicolas Prigent, Matteo Varvello, and Kyoungwon Suh, "Challenges in peer-to-peer gaming," *ACM SIGCOMM Computer Communication Review*, pp. 79-82, 2007.

[76] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid," *International Journal of Supercomputer Applications*, 2001.

[77] University of Chicago. (2008) The Globus Alliance. [Online]. http://www.globus.org/toolkit/

[78] Ian Foster, Car Kesselman, Jeffrey M. Nick, and Steven Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," p. 31, 2002.

[79] S. Tuecke et al., "Open Grid Services Infrastructure (OGSI)," 2003.

[80] Cristy Burne, "Policy - Grid computing walks the standard line: thinking inside the box," *International Science Grid This Week*, June 2008.

[81] Gregory B. Newby, Kevin Gamiel, and Nassib Nassar, "Secure information sharing and information retrieval infrastructure with GridIR.," in *Intelligence and Security Informatics: Proceedings of the First NSF/NIJ Symposium*, New York, 2003.

[82] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Alchemi: A.NET-based Grid computing framework and its integration into global grids," 2003.

[83] Martin Bichler and Kwei-Jay Lin, "Service-oriented computing," *Computer*, vol. 39, no. 3, pp. 99-101, Mar 2006.

[84] Eric A. Marks and Michael Bell, *Service Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology*.: John Wiley & Sons, Inc., 2006.

[85] Keith Bennett et al., "Service-based software: The future for flexible software," in *Seventh Asia-Pacific Software Engineering Conference*, Singapore, 2000, pp. 214-221.

[86] Galen Gruman and Eric Knorr. Infoworld. [Online]. http://www.infoworld.com/article/08/04/07/15FE-cloud-computing-reality_1.html

[87] Olaf Zimmermann, Vadim Doubrovski, Jonas Grundler, and Kerard Hogg, "Service-oriented architecture and business process choreography in an order management scenario: rationale, concepts, lessons learned," in *Conference on Object Oriented Programming Systems Languages and Applications*, San Diego, CA, 2005, pp. 301-312.

[88] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju, *Web Services - Concepts, Architectures and Applications*.: Springer, 2004.

[89] Tim Berners-Lee, *Weaving the Web: the original design of the World Wide Web by its inventor*.: HarperCollins, 2000.

[90] Roy T. Fielding and Richard N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115-150, May 2002.

[91] Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures," 2000.

[92] Frank Leymann, Dieter Roller, and Satish Thatte. (2003, August) Goals of the BPEL4WS Specification.

[93] OASIS. (2007, April) Web Services Business Process Execution Language Version 2.0. [Online]. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[94] Active Endpoints. Active Endpoints. [Online]. http://www.activevos.com/community-open-source.php

[95] The Apache Software Foundation. Apache ODE. [Online]. http://ode.apache.org/

[96] IBM. (2009) IBM. [Online]. www.ibm.com/software/integration/wps/

[97] Microsoft Corporation. (2009) Microsoft Corporation. [Online]. www.microsoft.com/biztalk

[98] Oracle. (2009) Oracle Technology Network. [Online]. http://www.oracle.com/technology/products/ias/bpel

[99] NetBeans. (2009) NetBeans. [Online]. http://enterprise.netbeans.org/

[100] The Eclipse Foundation. (2009) Eclipse.org. [Online]. http://www.eclipse.org/bpel/

[101] W3C. (2005, November) World Wide Web Consortium - Web Standards. [Online]. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/

[102] Workflow Management Coalition. (2009) Workflow Management Coalition. [Online]. http://www.wfmc.org/xpdl.html

[103] Keith Swenson. (2006, May) Thoughts on collaborative planning. [Online]. http://kswenson.wordpress.com/2006/05/26/bpmn-xpdl-and-bpel/

[104] Rachid Anane, Kuo-Ming Chao, and Yinsheng Li, "Hybrid Composition of Web Services and Grid Services," in *IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, Hong Kong, China, 2005, pp. 426-431.

[105] W3C. (2004, November) World Wide Web Consortium. [Online]. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

[106] Active Endpoints. (2008) ActiveBPEL Infocenter. [Online]. http://www.activebpel.org/infocenter/ActiveVOS/v50/index.jsp?topic=/com.activee.bpel.doc/html/UG22-1-2.html

[107] Cesare Pautasso, "BPEL for REST," in *7th International Conference on Business Process Management (BPM08)*, Milan, Italy, 2008, pp. 278-293.

[108] M.G. Nanda, S. Chandra, and V. Sarkar, "Decentralizing execution of composite Web Services," in *Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Vancouver, 2004, pp. 170-187.

[109] F. Montagut and R. Molva, "Enabling pervasive execution of workflows," in *International Conference on Collaborative Computing Networking, Applications and Worksharing*, San Jose, CA, EUA, 2005.

[110] Daniel Martin, Daniel Wutke, and Frank Leymann, "A novel approach to decentralized workflow enactment," in *12th International IEEE Enterprise Distributed Object Computing Conference*, Munich, Germany, 2008, pp. 127-136.

[111] R. Khalaf, O. Kopp, and F. Leymann, "Maintaining data dependencies across BPEL process fragments," *International Journal of Cooperative Information Systems, World Scientific Publishing*, vol. 17, no. 3, pp. 259-282, 2008.

[112] Cláudio Teixeira, Joaquim Sousa Pinto, and Joaquim Arnaldo Martins, "eABC: um repositório institucional virtual," in *ADIS Ibero-Americana : WWW/Internet 2005*, Lisbon, Portugal, 2005, pp. 653-656.

[113] Ex Libris. (2008) Ex Libris. [Online]. http://www.exlibrisgroup.com/category/Aleph

[114] Visual Resources Association. (2009, May) Visual Resources Association. [Online]. http://www.vraweb.org/resources/datastandards/vracore3/

[115] Moving Picture Experts Group. (2004) Moving Picture Experts Group. [Online]. http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm

[116] Joaquim S. Pinto, Joaquim A. Martins, Pedro Almeida, Marco Fernandes, and Helder Zagalo, "Portuguese Parliamentary Records: a Multimedia Digital Library Distributed Architecture, based on Web Services," in *NWeSP 2005: Next Generation Web Services Practices*, Seoul, South Korea, 2005, pp. 57-62.

[117] Pedro Almeida, Joaquim Arnaldo Martins, Joaquim Sousa Pinto, and Helder Troca Zagalo, "Audiovisual archive with MPEG-7 video description and XML database," in *ICEIS 2004: Sixth International Conference on Enterprise Information Systems*, Porto, Portugal, 2004, pp. 536-540.

[118] Microsoft Corporation. (2009) MSDN. [Online]. http://msdn.microsoft.com/en-us/library/ms689718%28VS.85%29.aspx

[119] Swish-e. (2007) Swish-e: Simple Web Indexing for Humans - Enhanced. [Online]. http://swish-e.org/

[120] QuiLogic Technologies. (2003) QuiLogic Technologies. [Online]. http://www.quilogic.cc/ifilter.htm

[121] Scott Oaks, Bernard Traversat, and Li Gong, *JXTA in a nutshell*.: O'Reilly Media, 2002.

[122] Marco Pereira, Tecnologia peer-to-peer para bibliotecas digitais [masters dissertation], 2008.

[123] OASIS. (2006) ebXML - Enabling a global electronic market. [Online]. http://www.ebxml.org/

[124] OASIS. (2004, October) UDDI.org. [Online]. http://uddi.org/pubs/uddi_v3.htm

[125] Microsoft Corporation. (2003, February) Windows Server 2003. [Online]. http://www.microsoft.com/windowsserver2003/techinfo/overview/uddiscen.mspx

[126] Alexander Mintchev, "Interoperability among service registry implementations: is UDDI standard enough?," in *2008 IEEE International Conference on Web Services*, Beijing, 2008, pp. 724-731.

[127] UNSPSC. (2009) UNSPSC. [Online]. http://www.unspsc.org/

[128] eCl@ss e.V. (2009) eCl@ss, the international standard for the classification of products and services. [Online]. http://www.eclass.de/index.html?no=intro&svt=2&navid=3065

[129] Mark Richards, "Creating an effective SOA service taxonomy," *SOA World Magazine*, November 2008.

[130] W3C. (2005, November) World Wide Web Consortium. [Online]. http://www.w3.org/Submission/WSDL-S

[131] The Apache Software Foundation. (2005) WebServices - Axis. [Online]. http://ws.apache.org/axis

[132] Marco Fernandes, Joaquim A. Martins, Joaquim S. Pinto, and Pedro Almeida, "Search engines evaluation for P2P based Digital Libraries," in *EATIS 2008: Euro American Conference on Telematics and Information Systems*, Aracaju, Brasil, 2008.

[133] Ronny Lempel and Shlomo Moran, "Predictive caching and prefetching of query results in search engines," in *12th International Conference on World Wide Web*, Budapest, Hungary, 2003, pp. 19-28.

[134] Indri. (2007) Indri. [Online]. http://www.lemurproject.org/indri

[135] The Apache Software Foundation. (2006) Apache Lucene. [Online]. http://lucene.apache.org/java/docs/

[136] University of Glasgow. (2007) Terrier Information Retrieval Platform. [Online]. http://ir.dcs.gla.ac.uk/terrier/

[137] Index Data. (2009) Index Data. [Online]. http://www.indexdata.com/zebra

[138] UMIC - Agência para a Sociedade do Conhecimento. (2010) RCAAP - Repositório Científico de Acesso Aberto de Portugal. [Online]. http://www.rcaap.pt/

[139] Eric Newcomer and Greg Lomow, *Understanding SOA with Web Services*.: Addison-Wesley, 2004.